2015

Formal methods paradigms for estimation and machine learning in dynamical systems

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

FORMAL METHODS PARADIGMS FOR ESTIMATION AND MACHINE LEARNING IN DYNAMICAL SYSTEMS

by

AUSTIN JONES

B.S., Washington University in Saint Louis, 2010
M.S., Washington University in Saint Louis, 2010

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2015

Approved by

First Reader

Calin Belta, Ph.D.
Professor of Systems Engineering
Professor of Mechanical Engineering
Professor of Bioinformatics

Second Reader

Mac Schwager, Ph.D.
Assistant Professor of Systems Engineering
Assistant Professor of Mechanical Engineering

Third Reader

Sean B. Andersson, Ph.D.
Associate Professor of Systems Engineering
Associate Professor of Mechanical Engineering

Fourth Reader

Christos G. Cassandras, Ph.D.
Professor of Systems Engineering
Professor of Electrical and Computer Engineering

Fifth Reader

John Baillieul, Ph.D.
Distinguished Professor of Systems Engineering
Distinguished Professor of Mechanical Engineering
Distinguished Professor of Electrical and Computer Engineering

Acknowledgments

This dissertation exists thanks to the extensive guidance from both of my advisors, Calin Belta and Mac Schwager. None of the results in this document would exist without their constant guidance and support, and the extra effort that joint advising requires. This dissertation reflects many hours of intense discussion at whiteboards and over skype, and many heavily annotated paper drafts.

I would like to thank my dissertation committee members for taking time from their busy schedules to review my thesis and attend my defense. I thank all of the members of HyNeSs, MSL, and GRITS labs, past and present, for their support and feedback over the years. In particular, I'd like to thank Zhaodan Kong and Derya Aksaray for steering my research in the direction of machine learning. The experiments in this paper would not have been possible without the aid and advice of Igor Cizelj, Eric Cristofalo, Dingjiang Zhou, Kevin Leahy, Teddy Ni, and Maria Svorenova. I thank Ebru Aydin-Gol for introducing me to signal temporal logic and Cristian Vasile for introducing me to sample-based techniques.

I would like to thank the Office of Naval Research, the National Science Foundation, and Boston University for funding my education and research.

I'd like to thank Dr. Magnus Egerstedt, without whose hospitality I would have had to spend most of the past year as a hermit in my apartment. I owe a debt of gratitude to Dana Sinno, Sung-Hyun Son, Jessica Stigile, Randy Paffenroth, Ben Slocumb, and Aubrey Poore for taking the time and energy to guide me through my (brief) stints in industry.

None of this would have been possible without the support of my parents, Kent and Terri, who have both encouraged my education since pre-school, or my in-laws, Mark and Ellen, whose hospitality has allowed me to live in Boston while I finished my degree. I would also like to acknowledge my grandparents Gerald, Joan, Sue, and

Jim, for their support of higher education. Above all, I want to thank my dear wife Elizabeth, whose multiple sacrifices, constant love and support, and tolerance of late nights has made this degree not only possible, but enjoyable.

FORMAL METHODS PARADIGMS FOR ESTIMATION AND MACHINE LEARNING IN DYNAMICAL SYSTEMS

AUSTIN JONES

Boston University, College of Engineering, 2015

Major Professors: Calin Belta
Professor of Systems Engineering
Professor of Mechanical Engineering
Professor of Bioinformatics

Mac Schwager
Assistant Professor of Systems Engineering
Assistant Professor of Mechanical Engineering

ABSTRACT

Formal methods are widely used in engineering to determine whether a system exhibits a certain property (verification) or to design controllers that are guaranteed to drive the system to achieve a certain property (synthesis). Most existing techniques require a large amount of accurate information about the system in order to be successful. The methods presented in this work can operate with significantly less prior information. In the domain of formal synthesis for robotics, the assumptions of perfect sensing and perfect knowledge of system dynamics are unrealistic. To address this issue, we present control algorithms that use active estimation and reinforcement learning to mitigate the effects of uncertainty. In the domain of cyber-physical system analysis, we relax the assumption that the system model is known and identify system properties automatically from execution data.

First, we address the problem of planning the path of a robot under temporal

logic constraints (e.g. avoid obstacles and periodically visit a recharging station) while simultaneously minimizing the uncertainty about the state of an unknown feature of the environment (e.g. locations of fires after a natural disaster). We present synthesis algorithms and evaluate them via simulation and experiments with aerial robots. Second, we develop a new specification language for tasks that require gathering information about and interacting with a partially observable environment, e.g. "Maintain localization error below a certain level while also avoiding obstacles." Third, we consider learning temporal logic properties of a dynamical system from a finite set of system outputs. For example, given maritime surveillance data we wish to find the specification that corresponds only to those vessels that are deemed law-abiding. Algorithms for performing off-line supervised and unsupervised learning and on-line supervised learning are presented. Finally, we consider the case in which we want to steer a system with unknown dynamics to satisfy a given temporal logic specification. We present a novel reinforcement learning paradigm to solve this problem. Our procedure gives "partial credit" for executions that almost satisfy the specification, which can lead to faster convergence rates and produce better solutions when the specification is not satisfiable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this dissertation, we present a collection of methods that are used to discover and enforce high-level behaviors in dynamical systems under uncertain or limited information. An example of this kind of problem is controlling a robot with an unknown or partially known dynamical model to perform a complicated "pick and place task" such as "Ensure that the product at workstation A is delivered to workstation B and that the product at workstation C is delivered to workstation D while avoiding region E. Go to region F after all products are delivered." This task involves two problems that are independently challenging: controlling the robot without the aid of a dynamical model, and solving a navigation problem that involves temporally and logically interleaved constraints, i.e. both product placements have to happen (logical) and products have to be picked up before they are placed (temporal). The solution to this problem must adapt to new information about the robot's dynamics in a way that still ensures it satisfies the given task.

The field of formal methods is concerned with checking, enforcing, and finding high-level specifications in a wide array of systems (Baier and Katoen, 2008). Many of these techniques were originally applied to software models, but recently there has been much interest and success in applying them to models of physical systems. The key feature of formal methods techniques is that they give deterministic or probabilistic correctness guarantees, such as "An unsafe configuration will never be reached

under any set of inputs to the model." or "This control policy will drive the system to one of the desired goal regions with probability at least 0.9." These guarantees are inherently tied to the model of the system under consideration. One of the challenges with applying formal methods techniques to physical systems is that, in contrast to software systems, a complete, accurate model of the system is rarely known.

The fields of estimation (Simon, 2006; Cover and Thomas, 2006) and machine learning (Marsland, 2009) are concerned with discovering *a priori* unknown information. In estimation, the goal is to use noisy measurements of a quantity to determine its value. In particular, we are concerned with active estimation, in which some deciding agent has control over which measurements to take in order to improve the quality of the estimate. An example of active estimation is controlling the orientation of a camera in order to track a moving object. One challenge with active estimation techniques is reconciling the actions that are optimal to take to gain information with other requirements on the system, e.g. using the camera to track object A if object B should periodically be in its field-of-view.

In machine learning, the goal is to learn a model or its features from data. For example, in the well-known problem of image discrimination, the goal is not to learn whether or not a particular picture is an image of a human face, but rather to learn what separates pictures of human faces from other images. One of the challenges with machine learning, however, is that frequently some form of significant expert guidance is required for learning to be successful. For instance, in the case of image discrimination, it is difficult for machine learning algorithms to be successful without knowing that features such as eyes, noses, mouths, etc. are important.

In this dissertation, we present a collection of algorithms that combine techniques and principles from formal methods and estimation/machine learning to solve classes of problems that are beyond the scope of established methods in each field. In essence,

we combine the ability of formal methods to answer questions about complicated, high-level system behavior with the ability of estimation and learning to gain insight about a system through data. This combination is non-trivial, as formal methods techniques must be made to handle model uncertainty while estimation and machine learning techniques must be extended to handle complicated, history-dependent features.

In this body of work, we consider high-level specifications that are formulated as temporal logic (TL) formulae. Temporal logic is an extension of propositional (Boolean) logic (Section 2) or predicate logic (Section 5) that allows us to reason about how a system evolves over time. Many of these properties can naturally be expressed in plain English, such as "Visit a recharging region 1 or 2 infinitely many times and ensure that obstacles are always avoided." or "The velocity of the car remains above 45 mph until braking begins." In addition to having plain English interpretations, these logics have rigorous mathematical formulations that we exploit in our algorithms.

We address four separate problems in this dissertation. First, we examine the problem of controlling a robot with known dynamics to gather the maximum amount of information about some unknown feature while still satisfying constraints on its motion given as a temporal logic formula. An example of this kind of problem is steering a robot on the surface of Mars to ensure that it collects as much scientifically-relevant information about soil while also avoiding hard-to-traverse terrain, ensuring that it visits locations where it will have a good link with the ground crew, and visiting locations where it can gather enough sunlight. This problem is challenging because each action the robot takes affects which parts of the environment its sensors can cover and its progress towards satisfying the given specification. We consider both finite- and infinite-horizon versions of this problem.

Second, we examine the case where, in contrast to the previous information gathering problem, the robot must not only estimate the state of some *a priori* unknown feature, but must also interact with it. An example of such a problem is steering a robot to find survivors after a natural disaster and guide any found survivors to a safe location. We develop a new temporal logic, called distribution temporal logic, that is capable of reasoning about the time evolution of not only the state of the system but also the uncertainty in this state. An example of a property that can be expressed in DTL is "Ensure that the uncertainty about the locations of fires remains below u for all time. If a survivor is found with probability p, guide the survivor to a safe location."

Third, we investigate the problem where, instead of steering a system to satisfy a given specification, we solve an inverse problem in which we want to determine what specification a given system is satisfying. In particular, we are interested in learning specifications that systems behaving in a desirable manner satisfy but those which are behaving in an undesirable manner do not. For example, in the context of naval surveillance, we learn specifications that law-abiding vessels are following and which law-avoiding vessels are not. We address this problem with off-line supervised learning, i.e. we infer a specification from a database of execution data labeled according to whether or not it represents desirable behavior, on-line supervised learning, i.e. instead of a database, we learn the formula as more data becomes available over time, and off-line unsupervised learning, i.e. the data in the database is unlabeled.

Finally, we consider the problem where again we are steering a system to satisfy a given specification, but we do not know what the dynamical model of the system is. The "pick and place" task is an example of this problem. We solve this problem via reinforcement learning, a machine learning technique in which the system takes actions, observes outcomes, and reinforces favorable actions. Over time, reinforcement

learning generates an optimal policy. In our case, we reinforce policies based on how close the outputs are to satisfying the given specification. This problem is not readily addressed with standard reinforcement learning methods, as the progress towards satisfaction depends on the history of the system, while standard methods assume stationarity.

1.1 Synthesizing Information Gathering Policies with Temporal Logic Constraints

In Chapter 3, we address the problem of controlling a robot with known dynamics and noisy sensors to gather as much information as possible about some *a priori* unknown feature while satisfying a high-level temporal logic mission specification. An example of this problem is using an autonomous underwater vehicle to measure the temperature of the ocean water as precisely as possible while avoiding high-traffic or tumultuous areas, periodically visiting locations where it can communicate with researchers, and eventually returning to a base station for recharging and servicing. We consider both finite-horizon (Jones et al., 2013a; Jones et al., 2015b) and infinite-horizon (Jones et al., 2015c) versions of this problem. In the infinite-horizon version, the constraints are given as linear temporal logic (LTL) formulae (Chapter 2). LTL is a well-known paradigm in formal methods (Kloetzer and Belta, 2008; Baier and Katoen, 2008), and can be used to describe many properties that can be stated in plain English, such as "Visit regions A and C or D infinitely often and ensure that region A is visited before C or D and always avoid region E." In the finite-horizon case constraints are given as a subset of LTL called syntactically co-safe LTL (scLTL) that is appropriate for describing constraints that can be satisfied in finite time (Kupferman and Vardi, 2001).

We consider a robot using a recursive Bayesian filter to estimate an uncertain quantity of interest (e.g. the locations of survivors) over a discrete environment. We take information-theoretic measures, namely the Shannon entropy (Shannon, 1948) of the Bayesian estimate in the finite-horizon case and the rate of this entropy in the infinite-horizon case, as a cost to be minimized over its closed-loop control policy. Our key insight is that we can construct a Markov decision process (MDP) that simultaneously encapsulates the robot's motion, its estimation process, and its progress towards satisfying a given TL formula. In this way we reduce our problem to the optimal control of an MDP.

For the finite-horizon case, we then use stochastic dynamic programming to find a feedback policy that globally optimally minimizes the uncertainty in the estimation task subject to satisfying the TL specification. We develop receding-horizon dynamic programing algorithms that a robot uses to generate a sub-optimal policy on-line. This approach drastically decreases computation time compared to the optimal dynamic programming solution. In our case studies, we have observed that the performance penalty we pay for this computational efficiency is small. We characterize these algorithms with theoretical guarantees and verify their performance in Monte Carlo simulations. We assess the algorithms' practicality in experiments with an aerial robot using an on-board camera to localize a randomly moving target.

We approach the infinite-horizon problem with a hierarchal planner. The low-level planner extends the receding horizon algorithm from (Jones et al., 2015b) to satisfy LTL constraints and maximize the mutual information rate (MIR) locally. The high-level planner uses a pre-computed policy to select inputs to the receding horizon algorithm that maximize the MIR over the long term. We construct this high-level policy using sampling-based techniques combined with value iteration. Our procedure is evaluated via Monte Carlo simulation.

In both sets of work, we synthesize high-level *mission* controllers rather than designing low-level *motion* controllers. We represent the environment as a discrete graph, where nodes are locations, rooms, or regions, and edges are available transitions between these regions. This implicitly assumes that our robot has low-level control and sensing capabilities to navigate and localize within its continuous world. Such low-level control and navigation can be accomplished with myriad existing techniques, for example with sophisticated SLAM algorithms (Thrun, 2008). For example, our algorithms consider controlling a high-level sequence of actions to ensure that data is uploaded and that recharging stations are visited, but we do not consider how to control wheel speeds to drive through a doorway given range measurements. The strength of our formulation is that it assumes a generic representation such that our procedures will work over a wide range of low-level motion control and data processing tools.

Our work addressing the finite-horizon problem is closely related to sensor placement (Akyildiz et al., 2002), where optimal locations are found for a static sensor network, and informative path planning, in which paths for mobile sensors are planned to collect a maximum amount of information. Previous works have focused on optimizing various information measures for both these problems, including Shannon mutual information and Fischer information. Many sophisticated algorithms exist for performing informative path planning in discretized environments for single or multiple agents using e.g. recursive greedy planning (Singh et al., 2007) or sequential dynamic programming (Meliou et al., 2007). Other constraints on the agents' motion such as communication constraints (Ghaffarkhah and Mostofi, 2012; Kassir et al., 2012), collision avoidance (Binney et al., 2010; Gan et al., 2012; Candido and Hutchinson, 2011), and environmental hazards (Schwager et al., 2011) have also been considered. Our use of scLTL constraints subsumes these more traditional kinds of constraints,

and allows for the expression of a much richer class of constraints. Further we provide *feedback policies*, not open-loop planned paths. In a stochastic iterative problem such as information gathering, it is well known that optimal policies must be feedback policies (Bertsekas, 2000; Bellman, 1957).

There exist forward computed receding horizon algorithms that are used to maximize local reward gathering while guaranteeing satisfaction of TL constraints (Wongpiromsarn et al., 2010; Ding et al., 2012). In contrast to these two works, our algorithm gives a dynamic programming receding horizon policy which is advantageous for optimizing stochastic objective functions such as information. Secondly, the distribution of the information-dependent cast that is incurred when taking an action from a particular state under our approach depends on what measurements and actions have been taken before. In contrast, the cost distribution is stationary in the cited works. Further, these works involved optimizing to satisfy the mission as quickly as possible, which can conflict with the goal of gathering as much information as possible. The receding algorithm presented in (Jones et al., 2015b) instead uses a new approach to ensure mission satisfaction within a certain time-bound. This increases the set of actions that the agent can take when compared with the standard approaches, which means that it can possibly take more informative paths. This change to the receding-horizon method is of independent interest as it may be adapted to other optimization problems with TL constraints.

The infinite-horizon problem is closely related to the problem of persistent monitoring, i.e. ensuring that the robot maintains a high-quality estimate indefinitely, under LTL constraints. Persistent monitoring strategies have recently been considered in the literature. In (Smith et al., 2012), the authors demonstrate how to regulate the speed of an agent moving along a fixed path such that the uncertainty about the state of the environment remains below a certain threshold indefinitely. This work

is extended to planning trajectories in (Lan and Schwager, 2013). The problem of multi-agent persistent monitoring is investigated in (Cassandras and Lin, 2013). The above works consider a measurement model that is independent of the agent's location in the environment and either completely unconstrained motion or motion along predefined paths. In contrast, we assume that the agent's sensing capability depends on its position in the environment and we incorporate LTL motion constraints. The most relevant existing work in this area is (Kress-Gazit et al., 2007), in which a related temporal logic (GR(1)) was used to encode tasks that were related to the agent's sensors. This work showed how to synthesize motion policies that were reactive to changes in the environment that were detected by the sensors, but no active estimation was performed to attempt to increase the knowledge of the unknown state of the underlying environment. In the search-and-rescue case study presented in(Kress-Gazit et al., 2007), the agents patrolled fixed paths until the survivors were "found", while in our approach, an agent would update its planned path on-line in order to localize the survivors as quickly as possible.

We presented results on this topic in (Jones et al., 2013b; Jones et al., 2015b; Jones et al., 2015c). In summary, our contributions are

- We defined and formalized the problem of gathering information under temporal logic constraints. (Jones et al., 2013b).
- We constructed a model that encapsulates how each action the robot takes affects the quality of its estimate and its progress towards satisfying a given TL specification (Jones et al., 2015b)
- We developed a pair of algorithms, a provably optimal algorithm and a sub-optimal algorithm with much better computational performance, that use this model to solve the constrained information gathering problem for finite-horizon

missions (Jones et al., 2015b). Both algorithms are guaranteed to satisfy the given specification.

- We extended these results to the problem of persistent monitoring and established a high-level planner that mitigates the myopia inherent in receding-horizon methods. (Jones et al., 2015a)

- While our algorithms have focused on information gathering, they are valid regardless of the chosen stochastic objective. This contributes to the field of stochastic optimization under temporal logic by augmenting the class of problems that can be addressed to include non-stationary cost distributions.

1.2 Temporal Logic Tasks under Uncertainty

In chapter 4, we develop a new paradigm for the specification of problems involving partially observable systems. Partially observable systems are doubly-embedded stochastic processes whose state can only be observed via an external related process. These models are frequently used in engineering to describe systems that evolve over time and observed via noisy measurements, such as a robot with noisy actuators that is operating in an environment and localizing itself via on-board LIDAR. In particular, we consider Partially Observable Markov Decision Processes (POMDPs) in which we update a Bayesian filter on-line to give a current probability distribution over the hidden state. This probability distribution is itself treated as a state, called a belief state in the POMDP literature.

In (Jones et al., 2013a), we defined distribution temporal logic (DTL), a predicate logic defined over sequences of belief states. DTL allows us to define properties not only of the evolution of the underlying unknown (hidden) state, but also the evolution

of the estimation process as well. With DTL, we can describe tasks such as "Measure the system state until estimate variance is less than v while minimizing the probability of entering a failure mode" or "If the most likely card to be drawn next is an Ace, increase your bet". DTL is a promising framework for high-level tasks over POMDPs as it can be used to describe the value of taking observations, as well as describing complex tasks defined over the hidden states of the system, and how to react to gains in certainty about the state of the system.

For stochastic systems with a fully observable state, some results exist that address the problems of model checking (determining with what probability any sample path satisfies a given specification), synthesis (synthesizing a policy that maximizes the probability of satisfying a given specification), and monitoring (determining on-line whether a system execution satisfies a given specification). For example, probabilistic computational tree logic (PCTL) can be used to describe temporal logic properties of Markov chains (Baier and Katoen, 2008). The probability of satisfaction of a TL formula over Markov chains can be calculated exactly using a reachability calculation (Baier and Katoen, 2008) or estimated from a subset of sample paths using statistical model checking (Sen et al., 2005; Zuliani et al., 2010). Formal synthesis for probabilistic robots modeled as Markov decision processes is also an active area of research (Lahijanian et al., 2012a; Ding et al., 2011; Wolff et al., 2012).

We introduces DTL as a means to formally pose formal methods problems over partially observable systems. We provide a statistical monitoring procedure that verifies *ex post facto* with what probability a particular execution of a POMDP satisfies a particular DTL specification.

Formal methods for stochastic systems with hidden states are difficult to develop. In general, if TL formulae are defined with respect to hidden states, their logical satisfaction can only be verified up to a probability based on partial knowledge of

the state. The logic POCTL* is an extension of PCTL that describes TL properties over hidden states and observations in Hidden Markov Models (HMMs) (Zhang, 2004; Zhang et al., 2005). HMMs are partially observable systems in which the true (hidden) state of the system evolves according to a Markov chain and can be probed by a related observation process. HMMs are widely used as models in speech recognition where phonemes are the hidden states of the system and vocal recordings are the observed states. POCTL* is used for checking properties such as "The probability that the sequence of hidden states $s^0 \dots s^t$ produces an observation sequence $o^0 \dots o^t$ is less than 0.1." The inclusion of observations in POCTL* allows us to specify some properties involving our external knowledge of the system, but, as we show in Chapter 4, DTL is more expressive and, for example, can be used to describe hypothesis testing tasks that POCTL* can't.

Recent development of point-based approximation methods (Smith and Simmons, 2005; Shani et al., 2012; Pineau et al., 2003; Kurniawati et al., 2008) and bisimulation - based reduction methods (Castro et al., 2009; Jansen et al., 2012) have made it possible to find sub-optimal solutions for maximizing the expected reward defined over hidden states in high-dimensional POMDPs with low computational overhead. It is well known, however, that maximizing the actual reward gathered in an execution of a POMDP is undecidable (Madani et al., 2003). Synthesizing policies over POMDPs to maximize the probability of satisfying a TL formula over hidden states is thus a hard problem, though some results exist for synthesis over short time horizons (Wongpiromsarn et al.,) and in systems where TL satisfaction can be guaranteed (Cimatti and Roveri, 2000).

The best action to take in a POMDP to increase the probability of satisfaction depends intimately on the quality of knowledge of the system. POCTL* describes the quality of knowledge based on the observation process, but this approach ignores

the richness of information conveyed by belief states. Information-theoretic measures defined over belief states can quantify the uncertainty (i.e. Shannon entropy) of the current estimate or the expected informativeness (i.e. mutual information) of future actions (Shannon, 1948; Cover and Thomas, 2006). Considering these two measures in mobile robots have increased environmental estimation quality (Julian et al., 2012; Bourgault et al., 2002; Hoffmann and Tomlin, 2010; Choi and How, 2010); incorporating them into TL-based planning for POMDPs will possibly yield similar results.

We presented results on this topic in (Jones et al., 2013a). In summary, our contributions are

- We develop a new type of temporal logic that can be used over partially observable systems. This logic expresses properties over the belief state, including measures of uncertainty.
- We define syntactically co-safe linear DTL (scLDTL), a DTL based on scLTL that can be used to prescribe finite-time temporal logic behaviors of POMDPs.
- We demonstrate that DTL can describe behaviors in partially observable systems that cannot be expressed by current TLs, including the logic POCTL* that was explicitly designed to be used in Hidden Markov Models.
- We provide an algorithmic procedure for evaluating the probability of satisfaction of an scLDTL formula with respect to an execution of a POMDP.

1.3 Learning Temporal Logic Specifications from Data

In Chapter 6, we address a trio of problems in the field of *specification mining* (or *temporal logic inference*). Specification mining is the "inverse" problem of model

checking. In model checking, the problem is to check whether a system satisfies a given specification via automata-theoretic (Baier and Katoen, 2008) or statistical methods (Zuliani et al., 2010). In specification mining, the problem is to determine which specification a system satisfies. We use specification mining to study systems with continuous state spaces with a special focus on security for cyber-physical systems. We approach these problems with principles from machine learning

Modern systems play increasingly important roles in the operation of critical infrastructure such as transportation networks and power grids. The large scale and highly nonlinear nature of some of these systems, however, renders the use of classical analysis tools difficult. Due to their networked nature, these systems are also vulnerable to external attacks or disruptions. Recent high-profile attacks, e.g. the Maroochy water breach (Slay and Miller, 2007) and the control system malware Stuxnet (Farwell and Rohozinski, 2011), highlight the need to understand system security (Fawzi et al., 2012; Gupta et al., 2010; Zhu and Martínez, 2013; Pasqualetti et al., 2011; Teixeira et al., 2012; Shoukry et al., 2013). In (Shoukry et al., 2013), a packet delay attack is considered, in which, by accessing and then delaying the sensor data, the adversary can harm the physical components of the system. In (Pasqualetti et al., 2011) and (Teixeira et al., 2012), more general issues such as detectability and identifiability for a wide range of attacks are investigated. In all the cited works, the physical system evolves according to a linear model that is known to the designers. This assumption is not consistent with the growing complexity of modern systems and the involvement of agents, such as humans, whose behaviors are generally quite hard to predict.

Anomaly detection is the problem of detecting patterns from data that do not conform to expected behavior. It has been used in a wide range of applications, such as intrusion detection for cyber-security, fault detection in safety-critical systems, and video surveillance of illicit activities (Chandola et al., 2009). Tools from statistics,

machine learning, data mining and information theory, such as k-means clustering and k-nearest neighbor (k-NN) graphs, have been adapted to solve the anomaly detection problem (Auslander et al., 2011; Kowalska and Peel, 2012). In general, existing techniques for anomaly detection infer a surface embedded in a high-dimensional feature space that separates normal and anomalous data. However, it is hard to interpret the meanings of the surfaces, especially in the context of prediction, knowledge base construction and on-line monitoring (i.e. determining on-line whether a behavior is anomalous) (Ustun et al., 2013).

In Chapter 6, we present model-free algorithms for system security. We use signal temporal logic (STL) formulae (Chapter 5), a specification language suited for continuous systems, as data classifiers. STL can be used to express system properties that include time bounds and bounds on physical system parameters, such as "If the boat remains in region A while maintaining its speed below 10 kph for 10 min., it is guaranteed to reach the port within 15 min." STL formulae resemble natural language, which means they can be useful for human operators. Further, the rigorous mathematical definition of the logic means that the formulae can be searched over using optimization methods and can be used in computational routines to automatically monitor systems for undesired behaviors. Specification mining in discrete systems is typically done via automata-based learning (Chen et al., 2013b; Angluin and Fisman, 2014), in which automata representations of LTL formulae are learned from finite substrings of infinite words. Since STL is not amenable to automata theory, we propose a novel machine learning-based approach for specification mining in systems with continuous state spaces.

We consider two critical security problems. The first is *anomaly learning* via supervised learning. In this case, given system outputs labeled according to whether a system behaves normally or not, we infer a temporal logic formula that can be

used to distinguish between normal system behaviors, e.g. the brakes of a train are engaged if the velocity is beyond a certain threshold, and anomalous (or undesired) behaviors, e.g. the brakes are not properly engaged due to attacks (Kong et al., 2014). The second problem is anomaly detection via unsupervised learning. In this case, the system outputs are not labeled, i.e. there is no expert-in-the-loop that determines whether a given trace represents normal or attacked operation, and we infer a formula to detect out-of-the-ordinary (anomalous) outputs (Jones et al., 2014).

Most of the recent research on logical inference has focused on the estimation of parameters associated with a given temporal logic structure (Asarin et al., 2012; Yang et al., 2012; Jin et al., 2013; Bartocci et al., 2013). That is, a designer gives a structure such as "The speed settles below v m/s within τ seconds" as an input and the inference procedure finds values for v and τ that optimize a given objective function. These structures are given as *parametric STL* (PSTL) formulae, an extension of STL in which bounds on space and time are replaced with variables. The given structure reflects domain knowledge of the designer and properties of interest to be queried. However, the selected formula may not reflect achievable behaviors (over-fitting) or may exclude fundamental behaviors of the system (under-fitting). Furthermore, by giving the formula structure *a priori*, the inference procedure cannot derive new knowledge from the data. Thus, in our procedures, we infer the formula structure (the form of the property that the system demonstrates) along with its optimal parameterization. We guide the search via the robustness degree (Chapter 5) a signed metric on the signal space which quantifies to what degree a signal satisfies or violates a given formula.

In (Jones et al., 2014), we defined a new logic called *inference parametric signal temporal logic* (iPSTL). iPSTL is a generalization of reactive Parametric Signal Temporal Logic (rPSTL), which we defined in (Kong et al., 2014). iPSTL is expressive

enough to capture properties that are crucial to a wide range of applications, e.g. naval surveillance and train network monitoring. We showed in (Kong et al., 2014) that we are able to build a directed acyclic graph (DAG) for all iPSTL formulae. The formulae in the DAG are organized according to how general they are such that if φ_2 is a child of φ_1, then the property described by φ_1 implies the property described by φ_2. This result enables us to formulate both the anomaly learning and the anomaly detection problems as optimization problems whose objective functions involve the robustness degree. We solve the optimization problems by combining discrete search over the DAG and a continuous search over the parameters.

To combat computational complexity, we have also developed an on-line supervised learning algorithm (Kong et al., 2015). That is, we present an algorithm to modify the classifying formula as more data is collected over time rather than learn a formula from a large batch of data. This procedure has lower computational costs than the supervised learning procedure developed in (Kong et al., 2014), making it applicable to high dimensional systems producing large amounts of data. Further, our new procedure requires no *a priori* system data, meaning it is applicable to systems from which no prior outputs are available.

Results on this topic have appeared in (Kong et al., 2014; Jones et al., 2014; Kong et al., 2015). To summarize, our contributions to this field are:

- We proposed the use of temporal logic formulae as data classifiers and formalized the supervised learning problem of inferring a temporal logic formula from labeled data (Kong et al., 2014).

- We defined a fragment of parametric signal temporal logic, called iPSTL, that is expressive enough to describe many properties of interest (Kong et al., 2014; Jones et al., 2014).

- We show that the set of iPSTL formulae can be organized into a DAG based on their inclusivity (Kong et al., 2014).

- We developed a supervised learning algorithm that combines searching for formula structures over the DAG and searching for optimal parameter values using derivative-free continuous optimization methods (Kong et al., 2014).

- We defined the problem of anomaly detection as an unsupervised learning problem and extended the results from (Kong et al., 2014) to develop an algorithm to solve it (Jones et al., 2014).

- We developed an on-line supervised learning algorithm that is more computationally efficient (Kong et al., 2015).

1.4 Reinforcement Learning and Temporal Logic

In Chapter 7, we consider the problem of controlling a system with unknown, stochastic dynamics, i.e. a "black box", to achieve a complex, time-sensitive task given as an STL formula. An example is controlling a noisy aerial vehicle with partially known dynamics to visit a pre-specified set of regions in some desired order while avoiding hazardous areas. When a stochastic dynamical model is known, there exist algorithms to find control policies for maximizing the probability of achieving a given propositional TL specification (Luna et al., 2014; Lahijanian et al., 2015; Svorenova et al., 2014; Kamgarpour et al., 2011) by planning over stochastic abstractions (Julius and Pappas, 2009; Abate et al., 2011; Lahijanian et al., 2015). Synthesizing policies to satisfy STL formulae in both deterministic and non-deterministic settings has only recently been considered (Raman et al., 2014; Raman et al., 2015). However, only a handful of papers have considered the problem of enforcing TL specifications on

a system with unknown dynamics. Passive (Brazdil et al., 2014) and active (Sadigh et al., 2014; Fu and Topcu, 2014) reinforcement learning has been used to find a policy over finite models, i.e. MDPs, that maximizes the probability of satisfying a given linear temporal logic formula.

In (Jones et al., 2015a), in contrast to the above works on reinforcement learning which use propositional temporal logic, we use STL (Chapter 5). By considering a logic that can directly reason about continuous states and bounded time, our method can be applied to classes of systems with bounded, real-valued state spaces, as opposed to those systems that can be modeled by Markov decision processes with finite state spaces. Further, the robustness degree associated with STL quantifies how strongly a given sample path exhibits an STL property as a real number rather than just providing a *yes* or *no* answer. This robustness gives more fine-grained information about how a given policy performs over the real-valued state space than simply using its probability of satisfaction.

One of the difficulties in solving problems with TL formulae is the history-dependence of their satisfaction. For instance, if the specification requires visiting region A before region B, whether or not the system should steer towards region B depends on whether or not it has previously visited region A. For LTL formulae with time-abstract semantics, this history-dependence can be broken by translating the formula to a deterministic Rabin automaton (DRA) and composing it with the system model to produce a product automaton that automatically takes care of the history-dependent "book-keeping" (Ding et al., 2014; Sadigh et al., 2014). However, in the case of STL, such a construction is difficult due to the time-bounded semantics. We circumvent this problem by defining a fragment of STL such that the progress towards satisfaction is checked with some finite number τ of state measurements. We thus define an MDP, called the τ-MDP whose states correspond to the τ-step history

of the system. The inputs to the τ-MDP are a finite collection of control actions.

We use a reinforcement learning strategy called Q-learning (Tsitsiklis, 1994), to find a non-stationary optimal policy, i.e. a mapping from states of the τ-MDP and the current time to the next control action to be taken. to either maximize the probability of satisfying a given STL formula, or maximize the expected robustness with respect to the given STL formula. Due to the history-dependent nature of formula satisfaction, we cannot formulate either objective as a sum of stage rewards. Thus, we cannot use traditional Q learning techniques to solve these problems. To solve this problem, we developed novel Q-learning algorithms that are guaranteed to converge to the optimal policies (with respect to a given τ-MDP) of each problem.

We propose that maximizing expected robustness is typically more effective than maximizing probability of satisfaction. We prove that in certain cases, the policy that maximizes expected robustness is guaranteed to also maximize the probability of satisfaction. However, if the given specification is not satisfiable, the probability maximization will return an arbitrary policy, while the robustness maximization will return a policy that gets as close to satisfying the policy as possible. Finally, we demonstrate through simulation case studies that the policy that maximizes expected robustness in some cases gives better performance in terms of both probability of satisfaction and expected robustness when fewer training episodes are available. This phenomenon occurs because the continuous nature of the robustness degree means that the robustness maximization algorithm reinforces policies that produce trajectories that almost satisfy the given formula more than policies that produce trajectories that do not come close. On the other hand, the probability maximization algorithm would reinforce both policies equally. Thus, before observing a satisfying trajectory, the probability maximization algorithm executes a random search, while the robustness maximization performs a search that is analogous to stochastic gradient descent.

Results on this topic appear in (Jones et al., 2015a). In summary, our contributions to this field are

- We define the problem of using reinforcement learning to enforce STL specifications rather than LTL specifications.
- We defined a fragment of STL whose progress towards satisfaction can be determined with a finite number of samples.
- We define a finite-memory Markov Decision Process abstraction that allows us to search for finite, history-independent policies
- We map the problem of enforcing an STL formula to maximizing expected robustness
- We prove that in some cases, maximizing expected robustness subsumes maximizing probability of satisfaction
- We define novel, provably-convergent Q-learning algorithms to maximize probability of satisfaction or expected robustness.
- We show via simulation that the algorithm that maximizes robustness can perform better in terms of both probability of satisfaction and expected robustness for a small number of training samples.

1.5 Organization of Dissertation

This dissertation is outlined as follows. In Chapter 2, we introduce some mathematical preliminaries. In particular, we introduce some of the basics of propositional temporal logic and finite systems and give a brief overview of how these models are

used in robotic navigation. Chapter 3 presents our work on synthesizing policies for information gathering under temporal logic constraints. Our work on distribution temporal logic, our novel paradigm for describing temporal logic properties of partially observable systems, is presented in Chapter 4. In Chapter 5, we review the basic properties of signal temporal logic. Chapter 6 presents our machine learning approach to specification mining problems for cyber-physical system security. Our reinforcement learning algorithm for steering systems with unknown dynamics to satisfy a given STL specification is presented in Chapter 7. Finally, we present some conclusions and future research directions in Chapter 8.

Chapter 2

Temporal Logic and Finite Systems

In this section, we present some background on finite systems and temporal logic that are preliminaries for Chapters 3 and 4.

2.1 Notation

For sets A and B, 2^A denotes the power set of A, $A \times B$ is the Cartesian product of A and B, and $A^n = A \times A \times \ldots A$. We use the shorthand notation $x^{1:t}$ for a time-indexed sequence of states $x^1 \ldots x^t$ where x^i is the value of x at time i. The set of all finite and set of all infinite words over alphabet Σ are denoted by Σ^* and Σ^ω, respectively. We frequently use the indicator function I, where

$$I(p) = \begin{cases} 1 & p \text{ is true} \\ 0 & p \text{ is false} \end{cases} \tag{2.1}$$

where p is a proposition or predicate.

2.2 Finite Models

In this section, we define a set of finite models that can be described by graph-like representations.

Definition 2.1. A *(deterministic) transition system* (TS) (Baier and Katoen, 2008)

is a tuple $TS = (Q, q_0, Act, Trans, AP, L)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Act is a set of actions, $Trans \subseteq Q \times Act \times Q$ is a deterministic transition relation, AP is a set of atomic propositions, and $L : Q \to 2^{AP}$ is a labeling function of states to atomic propositions. A *run* of a weighted transition system is a sequence of states $q^0 q^1 \ldots \in Q^*$ (or Q^ω) such that $q^0 = q_0$, and $\exists a^i \in Act$ such that $(q^i, a^i, q^{i+1}) \in Trans \ \forall i = 0, 1, \ldots$. An *output trace* of a run $q^0 q^1 \ldots$ is a word $w = w^0 w^1 \ldots$ where $w^i = L(q^i)$. □

Definition 2.2. A *discrete time Markov Chain* (MC) is a tuple $MC = (S, s^0, P)$ where S is a set of states, s^0 is an initial state of the system, and $P : S \times S \to [0, 1]$ is a probabilistic transition relation such that the chain moves from state s to state s' with probability $P(s, s')$. □

Definition 2.3. A *discrete time Markov decision process* (MDP) is a tuple $MDP = (S, s^0, P, Act)$, where S, s^0 are as defined for a MC, Act is a set of actions, and $P : S \times Act \times S \to [0, 1]$ is a probabilistic transition relation such that taking action a drives MDP from state s to state s' with probability $P(s, a, s')$. We denote the set of actions a that can be taken at state s such that $\exists s' \in S$ with $P(s, a, s') > 0$ as $Act(s) \subseteq Act$. A *sample path* of an MDP is a sequence of states $s^0 s^1 \ldots s^\ell$ with $P(s^i, a, s^{i+1}) > 0$ for some $a \in Act(s^i)$ $\forall i = 0, \ldots, \ell$. □

2.2.1 Stochastic Dynamic Programming

In this section, we give some basic preliminaries for stochastic dynamic programming. In stochastic dynamic programming, the goal is to select inputs to an MDP such that in expectation the resulting sample path minimizes a cost function defined over S. The *terminal cost* of a trace of length $\ell + 1$ is a function $g_\ell : S \to \mathbb{R}$ where $g_\ell(s)$ is the cost of ending a trace in state s. A discrete *stochastic optimal control problem* with

a *budget* of ℓ actions is an optimization of the form

$$\begin{gathered} \min_{a^{0:\ell-1}} E[g_\ell(s^\ell)] \\ \text{subject to} \\ a^i \in Act(s^i)\ \forall i = 0, \ldots, \ell-1 \\ s^{i+1} \sim P(s^i, a^i, s^{i+1}), \end{gathered} \tag{2.2}$$

where a^i is the action taken at time i. A *feedback policy* is a mapping $\mu : \mathbb{N} \times S \to Act$ such that $\mu(i, s^i)$ is the action taken at time i in state s^i. The solution to (2.2) is an *optimal feedback policy* μ^* such that

$$\mu^*(i, s) = \arg\min_{a^i \in Act(s)} E[J_{\ell-i-1}(s^{\ell-i-1})], \tag{2.3}$$

where

$$\begin{gathered} J_k(s) = \min_{a^{\ell-k:\ell}} E[g_\ell(s^\ell)] \\ \text{subject to} \\ s^{\ell-k} = s \\ a^i \in Act(s^i)\ \forall i = \ell-k, \ldots, \ell-1 \\ s^{i+1} \sim P(s^i, a^i, s^{i+1}), \end{gathered} \tag{2.4}$$

is the *cost to go function* with k steps to go. The cost to go and optimal policy are calculated via the following recursive relations, called the *Bellman iteration* (Bertsekas, 2000).

$$\begin{aligned} &J_0(s) = g_\ell(s) \\ &J_k(s) = \min_{a \in Act(s)} E[J_{k-1}(s^{\ell-k+1})] \\ &\mu^*(\ell-k, s) = \arg\min_{a \in Act(s)} E[J_{k-1}(s^{\ell-k+1})], \end{aligned} \tag{2.5}$$

for $k = 1, \ldots, \ell$.

2.3 Propositional Temporal Logic

In this section, we define the basics of some propositional temporal logics.

Definition 2.4. *Propositional logic* is a decidable logic defined over a set of atomic

propositions (variables that are either "true" or "false"). The set of propositional formulae is defined by the following four rules (Baier and Katoen, 2008)

1. True (denoted $\top$) is a propositional formula
2. Any proposition $p \in AP$ is a formula. This is the smallest possible formula, hence the adjective "atomic".
3. Given two formulae ϕ_1 and ϕ_2, $\neg\phi_1$, ("not ϕ_1") and $\phi_1 \wedge \phi_2$ ("ϕ_1 and ϕ_2") are also formulae.
4. Nothing else is a propositional formula.

□

An example of a propositional logic formula is $a \wedge b$, which is true iff the propositions a and b are both true.

In this dissertation, we use *temporal logics* to specify temporally layered properties of a variety of systems. Temporal logic is an extension of decidable logics to have "temporal modalities". That is, while typical logics can be evaluated over a "state", e.g. set of propositions, at a particular time, a temporal logic is decidable over states that evolve over time. In this thesis, we use exclusively *linear-time logics* , that is logics that are decidable over linear (as opposed to branching) sequences.

A logic is defined by its *syntax* , which is a set of rules about how formulae can be constructed using Boolean connectives, temporal operators, and atomic units, e.g. atomic propositions, and its *semantics* , which is a set of rules that tells how to interpret the logic over a set of models. In some cases, two formulae ϕ_1 and ϕ_2 may have differing syntax, but may be satisfied by the same set of models. In this case, we say that the two formulae are *semantically equivalent* , denoted $\phi_1 \equiv \phi_2$.

Example 2.1. The *syntax* of propositional logic is

$$\phi := \top | p | \neg\phi | \phi_1 \wedge \phi_2, \tag{2.6}$$

where p is an atomic proposition and ϕ, ϕ_1, and ϕ_2 are propositional formulae. Expression 2.6 expresses the rules given in Definition 2.4. The semantics of propositional logic is interpreted over sets $P \in 2^{AP}$. We say that P *satisfies* a propositional formula ϕ, denoted $P \models \phi$, if ϕ is true when interpreted over P. The semantics of propositional logic is defined recursively as

$$\begin{array}{lll} P \models \top & & \\ P \models p & \Leftrightarrow & p \in P \\ P \models \neg\phi & \Leftrightarrow & P \not\models \phi (\ P \text{ does not satisfy } \phi) \\ P \models \phi_1 \wedge \phi_2 & \Leftrightarrow & P \models \phi_1 \text{ and } P \models \phi_2 \end{array} \tag{2.7}$$

□

Now we define a generic linear time propositional logic called Linear Temporal Logic (LTL) that is frequently used to describe properties of discrete systems.

Definition 2.5 (LTL)**.** The *syntax* of linear temporal logic is defined as

$$\phi := \top | p | \neg\phi | \phi_1 \wedge \phi_2 | \bigcirc \phi | \phi_1 \mathcal{U} \phi_2 \tag{2.8}$$

where $p, \neg, \phi, \phi_1$, and ϕ_2 are as defined in (2.6) and $\bigcirc$ and $\mathcal{U}$ are the temporal operators "next" and "until", respectively. The following derived symbols are frequently

used for convenience.

$$\begin{aligned}
&\bot \equiv \neg\top \\
&\phi_1 \vee \phi_2 \equiv \neg(\phi_1 \wedge \phi_2) \\
&\phi_1 \Rightarrow \phi_2 \equiv (\neg\phi_1) \vee \phi_2 \\
&\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1) \\
&\diamondsuit\ \phi = \top \mathcal{U} \phi \\
&\square\ \phi = \phi \mathcal{U} \bot
\end{aligned} \tag{2.9}$$

The symbol $\bot$ is read as "false", $\vee$ as "or", $\Rightarrow$ as "implies", $\Leftrightarrow$ as "is equivalent to", $\diamondsuit$ as "eventually", and $\square$ as "always". The semantics of LTL is interpreted over infinite words $w = w^0 w^1 w^2 \ldots$ in the set $(2^{AP})^\omega$ and is defined recursively as

$$\begin{aligned}
&w^i \models \top \\
&w^i \models p && \Leftrightarrow && p \in w^i \\
&w^i \models \neg p && \Leftrightarrow && p \notin w^i \\
&w^i \models \phi_1 \wedge \phi_2 && \Leftrightarrow && w^i \models \phi_1 \text{ and } w^i \models \phi_2 \\
&w^i \models \phi_1 \vee \phi_2 && \Leftrightarrow && w^i \models \phi_1 \text{ or } w^i \models \phi_2 \\
&w^i \models \bigcirc \phi && \Leftrightarrow && w^{i+1} \models \phi \\
&w^i \models \phi_1 \mathcal{U} \phi_2 && \Leftrightarrow && \exists j \geq i \text{ such that } w^j \models \phi_2 \text{ and } w^k \models \phi_1\ \forall i \leq k < j \\
&w^i \models\ \diamondsuit\ \phi && \Leftrightarrow && \exists j \geq i \text{ such that } w^j \models \phi. \\
&w^i \models\ \square\ \phi && \Leftrightarrow && \forall j \geq i, w^j \models \phi.
\end{aligned} \tag{2.10}$$

A word w *satisfies* ϕ, denoted $w \models \phi$, iff $w^0 \models \phi$. We call the set of all words in $(2^{AP})^\omega$ that satisfy ϕ it's *language* , denoted $\mathcal{L}(\phi)$ □

Example 2.2. Consider the LTL formula

$$\phi =\ \square\ \neg\pi_1 \wedge (\ \square\ \ \diamondsuit\ \pi_2). \tag{2.11}$$

In plain English, (2.11) means "Never π_1 and always eventually (infinitely often) π_2." The word consisting of $\pi_3\pi_2$ repeated infinitely many times satisfies (2.11). The word $\pi_3\pi_2\pi_1 \ldots$ does not. □

In this work, we also consider linear time properties that can be satisfied with a

finite number of symbols. We model this set of properties with *syntactically co-safe* LTL (scLTL), a fragment of LTL. (Kupferman and Vardi, 2001; Latvala, 2003). A *fragment* of a logic is a logic that has the same semantics but whose syntax is a subset of the original.

Definition 2.6 (scLTL)**.** The *syntax* of scLTL is inductively defined as (Kupferman and Vardi, 2001):

$$\phi := p|\neg p|\phi \vee \phi|\phi \wedge \phi|\phi \mathcal{U} \phi|\bigcirc \phi|\diamondsuit \phi, \tag{2.12}$$

□

Any linear time property that can be checked with a finite prefix can be expressed as an scLTL formula (Kupferman and Vardi, 2001; Latvala, 2003). Technically, scLTL formulae are defined over infinite words, i.e. words in $(2^{AP})^{\omega}$. However, any infinite word that contains a finite prefix that satisfies an scLTL satisfies the formula, so for the purpose of this paper, we model scLTL as being defined over only those finite prefixes, i.e. words in $(2^{AP})^{*}$.

Example 2.3. The formula in (2.11) is in LTL but not scLTL, as its satisfaction requires infinite repetitions of π_2. On the other hand,

$$\phi = \diamondsuit \pi_2 \wedge (\neg \pi_1 \mathcal{U} \pi_2) \tag{2.13}$$

is in scLTL, as it only requires π_2 to occur once without π_1 occurring before π_2. That is, it requires no infinite repetitions or infinite-time safety conditions (hence the name "co-safe"). □

2.4 Automata

Many propositional temporal logics can be translated to an *automaton*, a graph- like representation that can be used to check whether a given sequence of inputs (words) satisfy a certain condition. In our case, this condition is a temporal logic formula. In this dissertation, we focus on LTL formulae that can be translated to a Buchi automaton (Baier and Katoen, 2008; Gastin and Oddoux, 2001).

Definition 2.7 (Buchi automaton)**.** A *deterministic Buchi automaton* (DBA) (Baier and Katoen, 2008) is a tuple $\mathcal{B} = (\Sigma, \Pi, \delta, \sigma_0, F)$ where Σ is a finite set of states, Π is a finite alphabet, $\delta \subseteq \Sigma \times \Pi \times \Sigma$ is a deterministic transition relation, $\sigma_0 \in \Sigma$ is the initial state of the automaton, and $F \subseteq \Sigma$ is a set of final (accepting) states. An *accepting run* on a DBA is a sequence of states in Σ^∞ that intersects with F infinitely often. The *language* of a DBA (written $\mathcal{L}(\mathcal{B})$) is the set of all accepting words in Σ^∞. □

Given an LTL formula ϕ, there exist algorithmic procedures (Bauer et al., 2011) to construct a DBA $\mathcal{B}_\phi$ with alphabet 2^{AP} such that the language of all words satisfying ϕ, $\mathcal{L}(\phi)$, is equal to $\mathcal{L}(\mathcal{B}_\phi)$. (Vardi, 1996). The details of this translation is beyond the scope of this dissertation. Such a DBA can be composed with a transition system and the resulting product can be used to check whether all paths in the TS satisfy ϕ (model checking) or to find an infinite path (that includes some sort of periodic behavior) that satisfies ϕ.

Definition 2.8 (Product Buchi automaton)**.** The *product Buchi automaton* of a transition system $TS = (Q, q_0, Act, Trans, AP, L)$ and a DBA $\mathcal{B}_\phi = (\Sigma, 2^{AP}, \delta, \sigma_0, F)$ is the Buchi automaton $\mathcal{P} = TS \times \mathcal{B}_\phi = (\Sigma_\mathcal{P}, \chi^0, Act, F_\mathcal{P}, \Delta_\mathcal{P})$ (Baier and Katoen, 2008). $\Sigma_\mathcal{P} \subseteq Q \times \Sigma$ is the state space of the automaton, $\chi^0 = (q_0, \sigma_0)$ is the initial state,

and $F_{\mathcal{P}} \subseteq Q \times F$ is the set of accepting states. The transition relation is defined as $\Delta_{\mathcal{P}} = \{(q,\sigma), p, (q',\sigma') | (q,p,q') \in Trans, (\sigma, L(q), \sigma') \in \delta\}$. The state of the automaton at time k, (q^k, σ^k) is denoted as χ^k for short. If $\chi^{0:k}$ satisfies the acceptance condition on $\mathcal{P}$, then $L(q^{0:k}) \models \phi$, where the labeling function L is applied element-wise. □

Frequently, we denote the state of a product automaton at time k, (q^k, σ^k), as χ^k for short. For scLTL, similar translations can be made to a different type of automaton, the finite state automaton (FSA) (Baier and Katoen, 2008; Kupferman and Vardi, 2001; Latvala, 2003).

Definition 2.9 (Finite State Automaton)**.** A *(deterministic) finite state automaton* (FSA) is a tuple $FSA = (\Sigma, \Pi, \Sigma_0, F, \Delta_{FSA})$ where Σ is a finite set of states, Π is an input alphabet, $\Sigma_0 \subseteq \Sigma$ is a set of initial states, $F \subseteq \Sigma$ is a set of final (accepting) states, and $\Delta_{FSA} \subseteq \Sigma \times \Pi \times \Sigma$ is a deterministic transition relation. An *accepting run* r_{FSA} of an automaton FSA on a finite word $\pi^0\pi^1 \ldots \pi^j \in \Pi^*$ is a sequence of states $\sigma^0\sigma^1 \ldots \sigma^{j+1}$ such that $\sigma^{j+1} \in F$ and $(\sigma^i, \pi^i, \sigma^{i+1}) \in \Delta_{FSA}$ $\forall i \in [0, j]$. We call the set of words that can lead to an accepting run on FSA the *language* of FSA, denoted $\mathcal{L}(FSA)$. □

Given an scLTL formula ϕ over the set of atomic propositions AP, there exist algorithms (Latvala, 2003) for creating an FSA with input alphabet 2^{AP} that accepts all and only words satisfying ϕ, i.e. an automaton FSA_ϕ such that $\mathcal{L}(FSA_\phi) = \mathcal{L}(\phi)$.

Definition 2.10 (Product FSA)**.** The *product automaton* between a deterministic transition system $TS = (Q, q_0, Act, Trans, AP, L)$ and an FSA $FSA_\phi = (\Sigma, 2^{AP}, \Sigma_0, F, \Delta_{FSA})$ is an FSA $\mathcal{P}_\phi = TS \times FSA_\phi = (\Sigma_{\mathcal{P}}, \chi^0, Act, F_{\mathcal{P}}, \Delta_{\mathcal{P}})$ (Baier and Katoen, 2008). $\Sigma_{\mathcal{P}} \subseteq Q \times \Sigma$ is the state space of the automaton, $\chi^0 = (q_0, \sigma_0)$ is the initial

state, and $F_{\mathcal{P}} \subseteq Q \times F$ is the set of accepting states. The transition relation is defined as $\Delta_{\mathcal{P}} = \{(q, \sigma), p, (q', \sigma') | (q, p, q') \in Trans, (\sigma, L(q), \sigma') \in \Delta_{FSA}\}$. □

The previous definitions are fairly standard tools in formal methods. In what follows, we present a set of tools that were used to enforce formula satisfaction in (Aydin Gol et al., 2012; Ding et al., 2012; Jones et al., 2013b; Jones et al., 2015b; Jones et al., 2015c).

Definition 2.11 (Distance to acceptance)**.** The *distance to acceptance* as a function $W : \Sigma_{\mathcal{P}} \to \mathbb{Z}^+$ such that $W(\chi)$ is the minimal number of actions that can be taken to drive $\mathcal{P}_\phi$ from χ to an accepting state in $F_{\mathcal{P}}$. If $\chi \in F_{\mathcal{P}}$, then $W(\chi) = 0$. If $W(\chi) = \infty$, then there doesn't exist an accepting run originating from χ. □

Definition 2.12 (k-step reachability neighborhood)**.** The *k-step boundary* about a state χ, denoted $\partial N(\chi, k)$ is the set of states that can be reached by applying exactly k inputs. □

2.5 Formal Methods and Robotics

In chapter 3, we consider constraints on the motion of robots that can be formulated as temporal logic formulae. These constraints subsume a large class of constraints that are commonly considered in the literature, such as reach-avoid specifications. In this section, we briefly give a "high-level" view of how formal methods may be used to solve robot navigation problems and motivate their use over *ad hoc* planning methods.

Example 2.4 (Reach-avoid)**.** Consider an agent in the environment shown in Figure 2·1(a). Each of the regions in the environment is either a goal region, an obstacle region, or empty. The goal of a "reach-avoid" problem is to reach a goal region

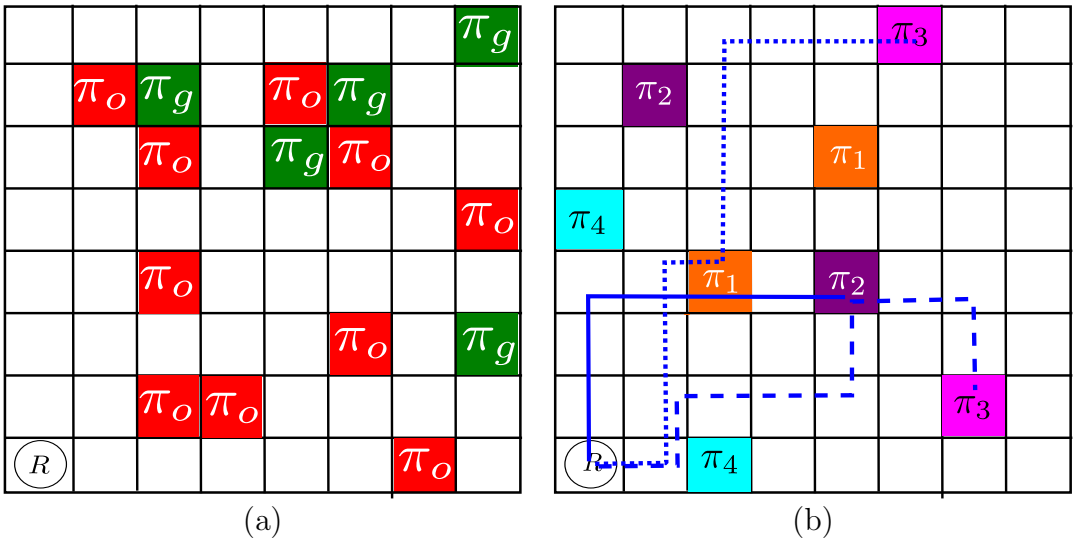


Figure 2·1: (a) An illustration of a "reach-avoid" problem in robotics. The agent must find a path to one of the green goal regions while avoiding all of the red obstacle regions. (b) An example of a specification that requires reactivity. The agent must conform to the specification 2.15

while avoiding all of the obstacle regions. This is typically achieved by applying a numerical reward to entering a goal region and a numerical penalty for entering an obstacle region and then using optimization methods such as dynamic programming to find a path from an initial state to a goal region that doesn't intersect any of the obstacle regions.

We can handle the same problem using formal methods by labeling each of the obstacle regions with the propositional label π_o (obstacle) and labeling each of the goal regions with the proposition π_g and enforcing the specification

$$\phi_{reachavoid} = \diamondsuit \pi_g \wedge (\neg \pi_g \mathcal{U} \pi_o), \tag{2.14}$$

which in plain English is "Eventually reach π_g and avoid π_o until π_g is reached."

□

In addition to typical navigation problems, temporal logic can be used to describe

more complex, temporally layered specifications than are easily modeled by using time-invariant rewards and costs on regions or states in the robot's environment. More importantly, the use of formal method allows us to find *automatically* the entire family of paths over the environment that satisfy the specification.

Example 2.5 (Reactive navigation)**.** Consider the example shown in Figure 2·1(b).

$$\begin{aligned}\phi_{reactive} = \quad & \phi_b(\pi_1, \pi_2, \pi_3) \vee \phi_b(\pi_2, \pi_3, \pi_4) \\ & \vee \phi_b(\pi_3, \pi_4, \pi_1) \vee \phi_b(\pi_4, \pi_1, \pi_2),\end{aligned} \tag{2.15}$$

where

$$\phi_b(\pi_j, \pi_k, \pi_m) = \ \diamondsuit\ (\pi_k \vee \pi_m) \wedge (\neg(\pi_k \wedge \pi_m)\mathcal{U}\pi_j)). \tag{2.16}$$

In plain English, $\phi_b(\pi_j, \pi_k, \pi_m)$ means "Region π_j is reached before either region π_k or region π_m." Thus, each instantiation of ϕ_b admits two *classes* of solutions ("π_j is reached and then π_k is reached" and "π_j is reached and then π_m is reached.") The formula $\phi_{reactive}$ is effectively a choice among four different instantiations of ϕ_b and hence admits eight total classes of solutions. Three example solutions from three different classes are shown in Figure 2·1(b). Note that the constraints of the problem depend explicitly on state history. That is, where the agent can go next depends explicitly on where the agent has been in the past. If the agent has visited π_1, visiting π_4 will not make progress towards satisfying $\phi_{reactive}$. However, if the agent has visited π_2, visiting π_4 will cause the specification to be satisfied.

Now, consider the case in which we want to find a path that satisfies the specification and is optimal in some sense, e.g. shortest distance or minimum energy. In the case of open-loop path planning, e.g. deterministic costs, it is possible to solve this problem by enumerating all eight solution classes, defining reachability constraints for each class by using traditional tools, solving the problem for all eight possible

solution classes, and picking the best solution from the eight possibilities. This process is tedious, as it requires the enumeration of all possible solution classes and the modeling and solution of eight different problems. Further, the enumeration of the different solution classes is prone to user error and grows more cumbersome as the number of possible solution classes grows.

If, on the other hand, we use formal synthesis methods to solve this problem, the only input the user has to provide is the temporal logic specification and the cost function. Given a formula, it is possible to construct *automatically* a graphical model (the product automaton) whose states correspond to the region of the environment and the progress towards satisfying the specification. In other words, the product automaton contains the minimal amount of "bookkeeping" required to describe all possible paths that can satisfy the specification. History-dependent properties become history independent when translated to the product automaton. The optimal control problem can be solved by applying Dijkstra's algorithm to generate the minimum cost path that is guaranteed to satisfy the specification. □

The usefulness of formal methods becomes even more evident when we consider closed-loop planning, i.e. optimizing with respect to stochastic costs. In general, we cannot use stochastic dynamic programming techniques over the original system to find an optimal policy that is guaranteed to satisfy the specification. The history dependence of specifications breaks the assumption of optimal substructures that is required to guarantee the optimality of dynamic programming (Bertsekas, 2000). In contrast, if we use formal methods, we can use stochastic dynamic programming to find a feedback policy over the states of the product automaton. This policy is guaranteed to satisfy the specification and to minimize the expected incurred cost. Again, the only user input required to this process is the specification and the cost

function.

Chapter 3

Informative Planning with Temporal Logic Constraints

In this chapter, we address the problem of constructing a control policy for a mobile agent to maximize the information it collects about an environment, while also satisfying a realistic high-level mission specification given as a temporal logic formula (Baier and Katoen, 2008). Consider a robot deployed to a building after a natural disaster to locate survivors. The robot's primary objective is to maximize the information it collects about the locations of survivors. However, while collecting information, it must satisfy realistic requirements, for example: "Avoid collisions with obstacles, visit data transmit stations after visiting areas of interest, and exit the building at one of several possible exits at the end of its tour, or if it becomes damaged while in operation." These complex specifications involving temporally and logically interleaved goals can be expressed naturally as a TL formula.

In Section 3.1, we present our models of the robot's motion and sensing capabilities and our model of the stochastically evolving environment. We present the problem of minimizing estimation uncertainty over a finite horizon under scLTL constraints in Section 3.2. We originally addressed this problem in (Jones et al., 2013b; Jones et al., 2015b). We consider an infinite-horizon version of this problem in Section 3.3. We originally addressed the infinite-horizon problem in (Jones et al., 2015c). Recently,

we have also considered a multi-agent version of this problem (Leahy et al., 2015), but this is not included in the dissertation.

3.1 Robot and Environment Models

In this section, we present the motion and sensing model of the robot and the model of the environment that we use throughout this chapter.

3.1.1 Motion model

We consider a single robot moving on a graph-like environment described by a transition system $\mathit{Robot} = (Q, q_0, \mathit{Act}, \mathit{Trans}, \mathit{AP}, L)$. Q is a finite set of states (the nodes of the graph) and q_0 is the robot's initial state. Act is a set of actions that the robot can enact. Trans is a transition relation (set of edges in the graph) such that $(q, a, q') \in \mathit{Trans}$ if action a drives *Robot* from state q to q'. AP is a set of atomic propositions (properties) and $L : Q \to 2^{\mathit{AP}}$ is the mapping from a state to the set of propositions it satisfies. We define a discrete clock k that is initialized to zero and increases by 1 each time *Robot* takes an action. We denote the *Robot*'s state at time k as q^k.

Remark 3.1. A transition system such as the one described above can be easily constructed by partitioning a planar or 3D environment. The states would correspond to the regions in the partition. The transitions and the corresponding actions would capture adjacency relations and feedback controllers driving all states from a region to another, respectively. Such controllers can be efficiently constructed for affine / multi-affine dynamics and simplicial / rectangular partitions (Belta et al., 2005; Belta and Habets, 2006). Such techniques can be extended to more complicated systems, such as unicycle and car-like dynamics through the use of input-output linearization

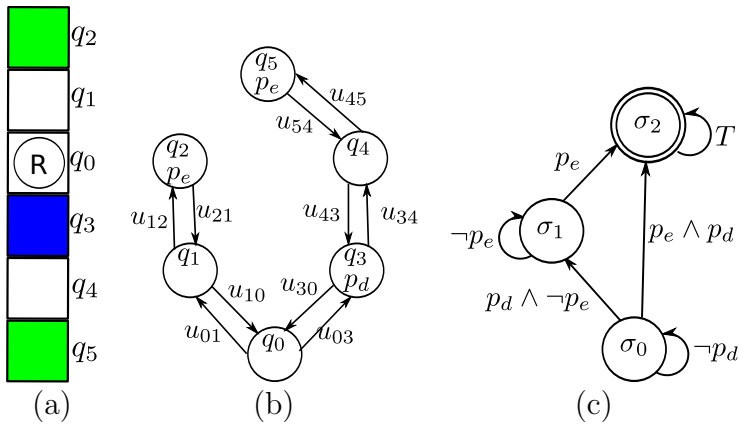


Figure 3·1: Illustration of Example 3.1. (a) The hallway environment in which the robot (denoted by R) operates. The data upload regions are blue and the exits are green. The labels q_i for the regions correspond to states in the transition system *Robot*. (b) Transition system constructed from the hallway environment. Action u_{ij} corresponds to a control driving the robot from region q_i to q_j. The labels p_d and p_e correspond to data upload and exit regions, respectively. (c) FSA constructed from scLTL formula ϕ_{ex} given in (3.5). Accepting states are indicated with double circles.

techniques. If a partition is not given *a priori* and the system has linear dynamics, there also exist techniques for iteratively partitioning the environment such that a controller is guaranteed to drive the system from any point in a region to a point in a neighboring region. (Aydin Gol et al., 2012). □

See Figure 3.1(a)-(b) for an example of a transition system corresponding to Example 3.1, a scenario that will serve as a running example throughout this paper.

Example 3.1. A ground robot equipped with a noisy camera is deployed to an office building after a natural disaster to locate survivors as precisely as possible. We consider a problem in which the robot operates in a hallway divided into six regions (Figure 3·1(a)). The robot must visit a region where it can upload data to rescue workers before eventually exiting the hallway. □

3.1.2 Sensing model

We associate with the environment a feature that evolves in time synchronously with the clock k according to the Markov chain $Env = (S, s^0, P)$. We denote the state of Env at time k as s^k. The initial state s^0 is *a priori* unknown.

At each time k, when the robot moves to state q^k, it measures s^k using noisy sensors. The sensor output at time k is a realization $y^k \in R_Y$ of a discrete random variable Y^k. In addition to q^k and s^k, the distribution of Y^k depends on the statistics of the sensor (how well the sensor measures the feature). We denote the time-invariant conditional measurement distribution as

$$h(y, s, q) = \begin{array}{l} Pr[\text{ the measurement is } y \\ |Env \text{ in state } s, Robot \text{ in state } q]. \end{array} \tag{3.1}$$

Example 3.1 (continued)**.** Let $S = \{0, 1\}^6$. The jth element of a state $s \in S$ corresponds to whether or not a survivor is located in region q_{j-1}, e.g. $s^k = [0, 1, 0, 0, 0, 0]$ means at time k a survivor is located in region q_1 and no other regions contain survivors (See Figure 3·1(a)). s^0 represents the initial locations of survivors, and P represents the probability that survivors move between neighboring regions, e.g. $P([0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])$ is the probability that a survivor moves from region q_1 to q_0. For simplicity, we assume that at most one survivor may be located in a region. The robot uses a camera to survey its surroundings and runs a detection algorithm on the gathered frames to estimate whether or not a survivor is located in its current hallway region, i.e. $y^k = 1$ if the algorithm produces a detection and $y^k = 0$ if it does not. □

The robot's estimate of s^k is given via the estimate probability mass function (pmf) b^k, called the *belief state* or *belief* , where $b^k(s) = Pr[s^k = s|y^{1:k}, q^{0:k}]$. The

belief state is initialized with a pmf b^0 that reflects the *a priori* belief about the value of s^0 and maintained via the Bayes filter

$$b^k(s) = \frac{h(y^k, s, q^k) \sum_{s' \in S} P(s', s) b^{k-1}(s')}{\sum_{\sigma \in S} h(y^k, \sigma, q^k) \sum_{s' \in S} P(s', \sigma) b^{k-1}(s')}. \tag{3.2}$$

The belief b^k evolves over time according to the MDP $Est = (B, b^0, P_{est}, Q)$. Q and b^0 are as defined previously. P_{est} is the probabilistic transition relation such that if b' is the result of applying the Bayes filter (3.2) with measurement y collected in state q, then $P_{est}(b, q, b')$ is the total probability of observing y, i.e.

$$P_{est}(b, q, b') = \sum_{s_1, s_2 \in S^2} h(y, s_2, q) P(s_1, s_2) b(s_1), \tag{3.3}$$

where a was the action that drove *Robot* to q. B is the (countably infinite) set of all possible beliefs that can be outputs of computing the Bayes filter with initial belief b^0 with a run of *Robot* along with the measurements the robot takes at each state in the run.

Remark 3.2. Note that here we use Q, rather than Act, which is the set of inputs inputs that are typically used when constructing MDPs involving a mobile robot. The likelihood function h, and by extension the Bayes filter mapping (3.2), is a function of *Robot*'s state ($q^k \in Q$) and not the action that drove it to that state ($a^{k-1} \in Act$). Thus, P_{est} depends on q^k and not a^{k-1}. □

In general, the states and transitions of Est can be arranged in a tree structure with root b^0. The children of a node b in the tree are $\{b' | \exists q \in Q, P_{est}(b, q, b') > 0\}$. Because of this branching, Est is also referred to as the *belief tree* in the partially observable MDP (POMDP) literature (Kurniawati et al., 2011) (Section 4.1).

3.2 Finite-Horizon Constrained Information Gathering

We consider a specific family of constraints on the motion of the agent that can be described with syntactically co-safe linear temporal logic (scLTL) (Kupferman and Vardi, 2001; Latvala, 2003) (Section 2.3). These formulae can be used to express a large class of natural constraints on the motion of robots, such as "Eventually reach the target region while avoiding unsafe regions. Visit region A or B before going to the target region. If you enter region C, go immediately to region D."

Several important problems can be considered as examples of information gathering with scLTL constraints. A micro-aerial vehicle tasked with urban surveillance must avoid buildings, visit data fusion centers to upload data, and finally land at a base or charging station, while simultaneously ensuring that it accurately characterizes the scene of interest. Information gathering problems with complex constraints have previously been formulated and solved such as in (Binney et al., 2010), in which the authors construct policies for an autonomous underwater vehicle that avoids high traffic areas and periodically communicates with researchers. In this section, we and introduce the MDP that encapsulates sensing, motion, and progress towards satisfying a given formula in Section 3.2.1. We formalize the constrained information gathering problem in Section 3.2.2. We present a pair of stochastic dynamic programming procedures to solve this problem, an optimal approach that precisely solves the problem and a sub-optimal approach that solves the problem with lower computational complexity in Section 3.2.3. Finally, we evaluate our results with simulations and experiments in Section 3.2.4.

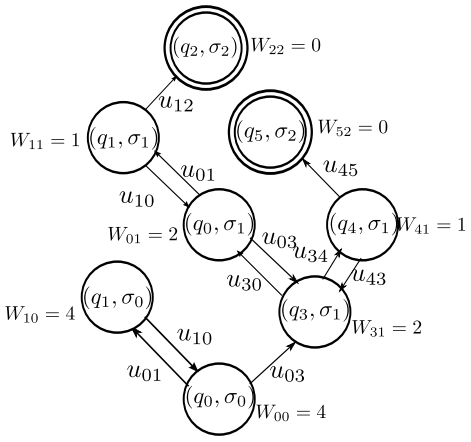


Figure 3·2: Product automaton constructed from the scenario in Example 3.1. Each state (q_i, σ_j) is annotated with its distance to acceptance, $W_{ij} = W((q_i, \sigma_j))$. Accepting states are denoted with double circles.

3.2.1 Full Model MDP

For a given scLTL formula ϕ, let $\mathcal{P}_\phi = \mathit{Robot} \times FSA_\phi$ as illustrated in Figure 3·2. Since the motion of an agent is deterministic, the correctness of a run of *Robot* with respect to ϕ can be determined precisely via $\mathcal{P}_\phi$, but its estimate b evolves probabilistically according to the MDP *Est*. As *Robot* and *Est* evolve synchronously, we can combine them into a single MDP that encapsulates the robot's movement and estimation process.

Definition 3.1. Full Model MDP

Consider a robot whose motion is modeled by *Robot* (Section 3.1.1) under a temporal logic constraint ϕ. Simultaneously, the robot is estimating the state of *Env* through the process *Est* (Section 3.1.2). The MDP $FullModel = (\Sigma_{\mathcal{P}_\phi} \times B, P_{tot}, Act, (\chi_0, b^0))$ describes the synchronous evolution of the robot's position and estimate pmf. The probabilistic transition relationship P_{tot} is defined as

$$P_{tot}((\chi, b), a, (\chi', b')) = P_{est}(b, q', b') I((\chi, a, \chi') \in \Delta_{\mathcal{P}_\phi}) \tag{3.4}$$

where I is the indicator function ($I(x \in X)$ is 1 if $x \in X$ and 0 if $x \notin X$) and $\chi' = (q', \sigma')$.

$FullModel$ has a tree structure with root (χ_0, b^0). The states at the i+1st level are the set of states that are reachable from taking one action and making one observation from any state in the ith level.

Example 3.1 (continued)**.** The regions where data can be uploaded are labeled with p_d and the regions with exits are labeled with p_e. The constraints on the motion of *Robot* may be expressed as

$$\phi_{ex} = \ \diamondsuit\, p_e \wedge (\neg p_e \mathcal{U} p_d), \tag{3.5}$$

meaning "Go to a data upload region (p_d) before going to an exit (p_e)." The corresponding finite state automaton $FSA_{\phi_{ex}}$ is shown in Figure 3·1 (c). Figure 3·3 shows the full model MDP constructed from the hallway scenario shown in Figure 3·1. To simplify the visual representation, edges in the MDP are grouped together via action, and the probabilistic transition relation is not explicitly shown. Note that any sequence of actions in this MDP will result in a state in which the automaton state component is accepting (denoted by double circles). The subscripts of the belief states correspond to states in the MDP Est (not shown) which were indexed sequentially as the tree was constructed, i.e. the subscript index increased by 1 each time a state was added to the topmost level of the tree. The temporal logic constraint (3.5) limits branching. This MDP has 218 total states, compared to 980 states in Est with depth 5. □

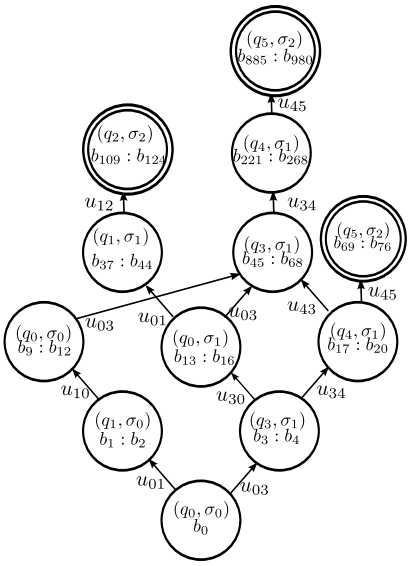


Figure 3·3: Full model MDP for hallway scenario (Example 3.1 with planning budget $\ell = 5$.

3.2.2 Problem Definition

We are interested in planning a finite trajectory for *Robot* such that the uncertainty about the state of *Env* is minimized when *Robot* completes the mission given by an scLTL formula ϕ over the labeled regions of the environment.

We quantify the uncertainty in a belief b with its Shannon entropy

$$H(b^k) = H(s^k | q^{0:k}, Y^{0:k}) = -\sum_{s \in S} b^k(s) \log_2 b^k(s) \tag{3.6}$$

In this paper, we measure the entropy in terms of *bits* . Roughly speaking, the Shannon entropy quantifies how hard it is on average to correctly guess the value of a random sample drawn from the given distribution. When entropy is measured in bits, then this can be interpreted as the number of yes/no inquiries one needs to make on average to guess the random value. If the mean of a Bernoulli random variable is 1, then its entropy is 0 bits, as we know that the outcome is guaranteed to be 1. If the mean is 0.5, then the entropy is 1 bit, as we need to ask the question "Is the

value 1?" in order to determine it precisely. By minimizing the Shannon entropy, we minimize how much additional information we would require to determine precisely the state of Env.

Our goal is to select actions $a^{0:t-1}$, $a^i \in Act \; \forall i \in 1, \ldots, t$, such that in expectation $H(b^t)$ is minimized. We predict the expected entropy of the belief that results from following a given path $q^{0:t}$ by calculating the conditional entropy $H(b^t|b^0, Y^{0:t}, q^{0:t})$, denoted $H(b^t)$ for short for short.

We allow *Robot* to take at most ℓ actions. This budget constraint reflects energy limitations on the robot. Each action consumes energy, and the robot should measure its environment and complete its mission (satisfy ϕ) before it drains its reserves.

The scLTL-constrained informative path planning problem is formulated in Problem 3.1.

Problem 3.1 (scLTL-constrained informative path planning)**.** Given a robot with model *Robot* operating in an environment Env, a finite budget ℓ, and an scLTL formula (mission) ϕ over AP, solve

$$\begin{gathered} \min_{a^{0:t-1}} E_{Y^{0:t}}[H(b^t|b^0, Y^{0:t}, q^{0:t})] \\ \text{subject to} \\ t \leq \ell \\ \phi \text{ is satisfied} \end{gathered} \tag{3.7}$$

□

We can cast Problem 3.1 as a constrained stochastic optimal control problem defined over $FullModel$.

Proposition 3.1. *Let $FullModel$ be defined according to Definition 3.1. Then, Problem 3.1 is equivalent to the following constrained stochastic optimal control problem:*

$$\min_{a^{0:t-1}} E_{Y^{0:t}}[H(b^t)] \quad \textit{subject to} \tag{3.8a}$$

$$t \leq \ell \tag{3.8b}$$

$$a^i \in Act((\chi^i, b^i)) \ \forall i = 0, \ldots, t-1 \tag{3.8c}$$

$$(\chi^{i+1}, b^{i+1}) \sim P_{tot}(\cdot, a^i, (\chi^i, b^i)) \tag{3.8d}$$

$$\chi^t \in F_{\mathcal{P}_\phi}. \tag{3.8e}$$

□

Remark 3.3. In our algorithms, we only consider a finite subset of $FullModel$, namely the states and transitions such that every sample path of length at most ℓ beginning from the initial state (χ_0, b^0) is guaranteed to satisfy ϕ. The number of states in this finite subset of $FullModel$ is less than or equal to the number of states in the first $\ell + 1$ levels of Est. This is due to the fact that the children of a node in Est correspond to all actions that can be taken in the current state. However, taking certain actions may violate ϕ or ensure that it cannot be satisfied within the remaining budget. The children of such actions do not appear in $FullModel$. □

Remark 3.4. The cost in (3.8) is defined with respect to a pmf, which may recall stochastic optimal control of POMDPs (Kaelbling et al., 1998) (Section 4.1). It is tempting to use established approximation methods such as point-based value iteration (Pineau, 2004; Shani et al., 2012) to optimize over the continuous space of pmfs rather than enumerate a finite number of states. However, in our formulation, the cost of a pmf $H(b)$ is highly non-linear in the components $\{b(s)|s \in S\}$. Since the cost-to-go function is not piecewise linear, such approximations cannot be applied

(Smallwood and Sondik, 1973). □

3.2.3 Algorithms

In this section, we connect Problem 3.1 to Markov decision processes (MDPs) and present two solutions: an off-line dynamic programming approach and an on-line receding-horizon dynamic programming approach. The off-line dynamic programming approach is outlined in Algorithm 3.1.

Algorithm 3.1 Returns a policy for ℓ time units that satisfies ϕ and minimizes $E_{Y^{0:t}} H(b^t)$

```
1: function FiniteHorizonDP(P_φ,χ^0,b^0,ℓ)
2: if W(χ^0) > ℓ then
3:    return None
4: automatonStates := PossibleStates(P_φ,χ^0,ℓ, ℓ)
5: (MDPStates,P_tot,actionSet) := ConstructMDP(P_φ, χ^0,b^0,automatonStates,ℓ)
6: return BellmanIteration(MDPStates,P_tot,actionSet)
```

First, Algorithm 3.1 ensures that the planning budget ℓ is long enough to allow *Robot* to satisfy ϕ, i.e. checks whether the distance to accepting $W(\chi^0)$ is at most ℓ. Next, Algorithm 3.2 calculates ℓ subsets of $\Sigma_{\mathcal{P}_\phi}$ automatonStates$[i]$, $i = 1, \ldots, \ell$. Each state $\chi \in$ automatonStates$[i]$ is reachable from χ^0 in exactly i transitions (is in $\partial N(\chi^0, i)$) and can reach an accepting state within the remaining budget of $\ell - i$ transitions ($W(\chi) \leq \ell - i$).[1]

Next, we use Algorithm 3.3 to build a finite subset of the infinite-dimensional MDP $FullModel$ such that under any admissible control policy, the trace of the constructed MDP results in a trace of *Robot* that satisfies ϕ. The root of the constructed MDP is (χ^0, b^0). The ith level of the tree is constructed from the $i-1$st level by enumerating for each state (χ, b) in the $i-1$st level the actions that can enable a transition from

[1] Algorithm 3.2 uses two separate inputs, m and ℓ, to determine the number of sets constructed and the threshold on $W(\chi)$, respectively. This may seem redundant as $m = \ell$ here, but this distinction is necessary for our approximation algorithm (Algorithm 3.4) presented in Section 3.2.3.

Algorithm 3.2 Calculate all product automaton states that can lead to accepting runs in ℓ or fewer transitions.

1: **function** PossibleStates($\mathcal{P}_\phi$,χ^0,m,ℓ)
2: automatonStates[0] := χ^0
3: **for** $i := 1$ to m **do**
4: automatonStates[i] = $\{\chi \in \Sigma_{\mathcal{P}_\phi} | W(\chi) \leq \ell - i\} \cap \partial N(\chi^0, i))$
5: **return** automatonStates

χ to a state $\chi' = (q', \sigma') \in$ automatonStates[i]. For each such action a, we enumerate all the possible measurements $y \in R_Y$ that can be observed in q' and calculate the estimate pmf b' that results from applying the Bayes filter (line 8) and add the state (χ', b') to the ith level. The probabilistic transition relation P_{tot} is then constructed (line 11).

Algorithm 3.3 Construct the full model MDP

1: **function** ConstructMDP($\mathcal{P}_\phi$,χ^0,b^0,automatonStates, ℓ)
2: MDPStates[0] := (χ^0, b^0)
3: **for** $j := 1$ to ℓ **do**
4: **for all** $(\chi, b) \in$ MDPStates[$j-1$],$\chi' \in$ automatonStates[j] **do**
5: **if** $\chi \notin F_{\mathcal{P}_\phi}$ **then**
6: $a :=$ action such that $(\chi, a, \chi') \in \Delta_{\mathcal{P}_\phi}$
7: **for all** $y \in R_Y$ **do**
8: $b' :=$ output of (3.2) with b, a, y
9: MDPStates[j].add((χ', b'))
10: actionSet[(χ, b)][$j-1$].add(a)
11: $P_{tot}((\chi, b), a, (\chi', b')) :=$ output of (3.4)
12: **return** (MDPStates,P,actionSet)

Finally, the function BellmanIteration uses standard, finite-horizon Bellman iteration to calculate the optimal policy with zero stage costs.

The reactive policy generated by Algorithm 3.1 can be applied to *FullModel* to generate trajectories of *Robot* that are guaranteed to satisfy ϕ and in expectation minimize the Shannon entropy of the resulting estimate. The next theorem establishes the optimality of this method.

Theorem 3.1. *The policy generated by Algorithm 3.1 is the exact solution of Problem 3.1.*

Proof. Let $\mu^*_{\chi^0,b^0,\ell,\ell}$ (μ^*, briefly) be the result of applying Algorithm 3.1 to $MDP_{\chi^0,b^0,\ell,\ell}$ (MDP_ℓ, briefly), the MDP constructed in Algorithm 3.3. μ^* is calculated by applying finite horizon Bellman iteration to MDP_ℓ, so, by principle of optimality,

$$\mu^* = \arg\min_{\mu \in M_\ell} E_{(MDP_\ell,\mu)}[H(b^t)], \tag{3.9}$$

where $M_{\chi^0,b^0,\ell,\ell}$ (M_ℓ) is the set of all possible policies of length ℓ that can be defined over MDP_ℓ. Thus in order to prove the optimality of μ^*, we must prove that the set of traces produced by MDP_ℓ under the policies in M_ℓ are exactly those traces which satisfy the constraints (3.8b)-(3.8e).

(3.8b) The max depth of MDP_ℓ is ℓ.

(3.8c) In order for an action a to be in actionSet$[(\chi, b)][i]$, (a) $(\chi, a, \chi') \in \Delta_{\mathcal{P}_\phi}$ and (b) $P_{est}(b, q', b') > 0$. (a) and (b) together imply $P_{tot}((\chi, b), a, (\chi', b')) > 0$, meaning (3.8c) is satisfied for all $a \in$ actionSet$[\cdot][i]$, $i = 0 \ldots \ell - 1$.

(3.8d) This follows immediately from (a) and (b) and Algorithm 3.3, line 11.

(3.8e) Since $\chi \in$ automatonStates$[i]$ $\forall(\chi, b) \in$MDPStates$[i]$, every state in MDPStates$[i]$ is at most $\ell - i$ transitions away from a state $(\chi_f, b_f) \in F_{\mathcal{P}_{phi}} \times B$. Every state $\chi \in$ automatonStates$[i]$ with distance to acceptance $W(\chi) < \ell - i$ has a neighbor that is closer to an accepting state, so $\exists \chi' \in$ automatonStates$[i+1]$ such that $W(\chi') < \ell - i - 1$. Every state in MDPStates$[i]$ is either an accepting state or has a child in MDPStates$[i+1]$. Since MDPStates$[\ell]$ is constructed from automatonStates$[\ell]$, either each state in MDPStates$[\ell]$ is accepting or MDPStates$[\ell] = \emptyset$, implying $\exists k < \ell$ such that every state in MDPStates$[k]$ is accepting. Therefore, every trace produced by MDP_ℓ must end in accepting state.

□

We expect the performance of the algorithm to improve with a longer budget. Indeed,

Proposition 3.2. *Increasing the budget ℓ cannot decrease and can increase the expected quality of the calculated optimal policy.*

Proof. Let $\ell_1 < \ell_2$ and let $\mu_i = \mu^*_{\chi^0,b^0,\ell_i,\ell_i}$ Since the states and actions in MDP_{ℓ_1} at each level are included in the states and actions in MDP_{ℓ_2} at the same level for the first ℓ_1 levels, $M_{\ell_1} \subseteq M_{\ell_2}$. Thus,

$$\min_{\mu \in M_{\ell_1}} E_{(MDP_{\ell_1},\mu)}[H(b^t)] \geq \min_{\mu \in M_{\ell_2}} E_{(MDP_{\ell_1},\mu)}[H(b^t)]. \tag{3.10}$$

Further, since the family of possible traces produced by MDP_{ℓ_1} under μ_2 is a subset of the family of possible traces produced by MDP_{ℓ_2} under μ_2,

$$E_{(MDP_{\ell_1},\mu_2)}[H(b^t)] \geq E_{(MDP_{\ell_2},\mu_2)}[H(b^t)]. \tag{3.11}$$

Inequalities (3.10) and (3.11) together imply the proposition. □

However, calculating longer policies is considerably more computationally expensive. More specifically,

Lemma 3.1. *The time complexity of Algorithm 3.1 is $O(|Act|^\ell |R_Y|^\ell |S|)$.*

The average case complexity can be lower than the bound given in Lemma 3.1 due to the pruning of actions that occurs when constructing $FullModel$ (Remark 3.3), but the exponential dependence on ℓ remains.

Receding Horizon Dynamic Programming

The computational cost of Algorithm 3.1 is too great for large budgets. As an alternative, Algorithm 3.4 constructs an approximation to the optimal policy on-line using a receding-horizon implementation. At each time step k, Algorithm 3.4 constructs an MDP whose root is the pair (χ^k, b^k) and whose depth is at most some finite horizon m (line 5). The ith level of each MDP is constructed from automaton state/pmf pairs that are reachable from (χ^k, b^k) in i transitions and can satisfy ϕ in $\ell - (k+i)$ actions. Bellman iteration is performed on this MDP to form the m-step optimal policy (Line 6). The agent applies the policy for $n \leq m$ time steps. Then, the agent begins the process again from an MDP with root (χ^{k+n}, b^{k+n}). This is repeated until the trajectory of *Robot* has satisfied ϕ.

Algorithm 3.4 Receding-horizon approximation to Algorithm 3.1.

```
1: function RecedingHorizonDP(P_φ,χ^0,b^0 ,m,n,ℓ)
2: χ := χ^0; b := b^0;k := 0
3: if W(χ^0) > ℓ) then
4:    return None
5: while χ ∉ F_{P_φ} do
6:    automatonStates := PossibleStates(P_φ,χ,m,ℓ − k)
7:    (MDPStates,P_tot,actionSet) := ConstructMDP(P_φ,χ,b,automatonStates)
8:    μ :=BellmanIteration(MDPStates,P_tot,actionSet)
9:    if k ≥ ℓ − m then
10:      n := ℓ − k
11:   for i := 1 to n do
12:      (χ, b) := result from applying μ(i, (χ, b))
13:      k + +
```

Lemma 3.3 states the time complexity of Algorithm 3.4, which leads to a significant savings in computational effort if ℓ is long.

Lemma 3.2. *The time complexity of Algorithm 3.4 with budget ℓ, planning horizon m, and action horizon n is $O(|Act|^m |R_Y|^m |S| \lceil \frac{\ell}{n} \rceil)$.*

Now, we must establish the correctness of Algorithm 3.4 with respect to the specification ϕ.

Theorem 3.2. *If $W(\chi^0) \leq \ell$, the solution of Algorithm 3.4 is guaranteed to satisfy ϕ.*

Proof. At time $k < \ell - m$, if the full state of the system is described by (χ^k, b^k), Algorithm 3.4 constructs $MDP_{\chi^k,b^k,\ell,m}$ (line 5). If we apply any policy $\mu \in M_{\chi^k,b^k,\ell,m}$ for n time periods, we are guaranteed to visit a sequence of states (χ^{k+i}, b^{k+i}) such that $W(\chi^{k+i}) \leq \ell - k - i \; i = 1, \ldots, n$. Thus while $k < \ell - m$, $W(\chi^k) \leq \ell - k$. If Algorithm 3.4 has not terminated when $k \geq \ell - m$, Algorithm 3.4 constructs $MDP_{\chi^k,b^k,\ell-k,\ell-k}$, and applies the policy μ^*, effectively executing Algorithm 3.1 with initial state (χ^k, b^k) and horizon $\ell - k$. Since applying Algorithm 3.4 up until time k guarantees $W(\chi^k) < \ell - k$, by Theorem 3.1, *Robot* will satisfy ϕ within $\ell - k$ steps. □

Finally, we have to consider performance. Calculating a precise sub-optimality gap between Algorithms 3.1 and 3.4 is difficult due to the non-linear natures of the entropy measure and the Bayes filter and variability of possible automaton structures. Instead, we present a pair of results that establishes some general properties of the solution quality of Algorithm 3.4.

Proposition 3.3. *If $\ell = m$, executing Algorithm 3.4 and 3.1 result in the same policy.*

Proof. If $\ell = m$, then both Algorithm 3.1 and Algorithm 3.4 construct the same MDP and calculate the same policy. $MDP_{\chi^0,b^0,\ell,\ell}$ (lines 5 and 5, resp.) and use Bellman iteration (lines and 6, resp.) to construct the optimal policy $\mu^*_{\chi^0,b^0,\ell,\ell}$. In Algorithm 3.1, lines 7-8 ensure that $\mu^*_{\chi^0,b^0,\ell,\ell}$ is applied for ℓ time steps. □

Corollary 3.1. *If $m < \ell$, the policies produced by Algorithm 3.4 are in general suboptimal.*

Proof. Let $\sharp$ be the policy concatenation operator such that for two policies μ_1, μ_2, $\mu_1[n_1]\sharp\mu_2[n_2]$ is the policy resulting from applying μ_1 for n_1 time steps and μ_2 for n_2 time steps. Also define $[\cdot], \sharp$ set wise such that if M_1, M_2 are sets of policies, then $M_1[n_1]\sharp M_2[n_2] = \{\mu_1[n_1]\sharp\mu_2[n_2] | \mu_1 \in M_1, \mu_2 \in M_2\}$. Let $M^{RH}_{\ell,m,n}$ be the family of policies considered in Algorithm 3.4. Then,

$$\begin{aligned} M^{RH}_{\ell,m,n} = \; & \{M_{\chi^0,b^0,\ell,m}[n]\sharp M_{\chi^n,b^n,\ell,m}[n]\sharp \\ & \ldots\sharp M_{\chi^{pn},b^{pn},\ell-pn,\ell-pn} \\ & |(\chi^{rn}, b^{rn}) \in MDP_{\chi^{(r-1)n},b^{(r-1)n},\ell,m} \\ & \forall r = 1, \ldots, p\} \end{aligned} \tag{3.12}$$

where $\ell - pn \leq m$. Thus, $\mu^* \in M^{RH}_{\chi^0,b^0,\ell,m,n}$, but $\mu^{RH}_{\chi^0,b^0,\ell,m,n} \neq \mu^*$ in general. Since $M^{RH}_{\chi^0,b^0,\ell,m,n} \subseteq M_\ell$, no other policy in $M^{RH}_{\chi^0,b^0,\ell,m,n}$ can outperform it. Therefore Algorithm 3.4 with $m < \ell$ is suboptimal.

□

Intuition tells us that the performance should depend on the planning horizon m and the action horizon n, but similarly, we cannot characterize this relationship precisely. Instead, we will observe the effects of varying each of these quantities empirically in the next section.

3.2.4 Case Study and Experiments

In this example, we use simulations and experiments to validate and compare Algorithms 3.1 and 3.4.

Conclusion of running example

We tested Algorithms 3.1 and 3.4 using a simulation of Example 3.1. We simulated 500 Monte Carlo trials of each method with horizon $\ell = 8$ on a machine with a 2.1 GHz processor and 7.4 GB RAM. Parallelization was not used. For the receding horizon method, the planning and acting horizons were $m = 3$, $n = 2$, respectively. The average terminal entropy $H(b^t)$ that resulted from the trajectories generated as well as the average mistake rate $m(t)$, defined as

$$m(t) = \sum_{j|q_j \in Q} | \sum_{\sigma|\sigma_j \neq s_j^t} b^t(\sigma) - s_j^t|, \tag{3.13}$$

for Algorithms 3.1 and 3.4 are given in Table 3.1.

That is, $m(t)$ is the sum over the rooms of the difference between the true state of the room (whether or not a survivor is present) and the belief about the room (probability that a survivor is present).

All trajectories generated by both algorithms satisfied the given scLTL specification. The resulting statistics for the simulation are given in Table 3.1. The calculation time T for Algorithm 3.1 was the total time required to calculate the optimal policy. For Algorithm 3.4, this is the average time required to calculate the optimal policy per trial. As expected, Algorithm 3.1 performed better (achieved lower average terminal entropy and error) than Algorithm 3.4. The difference in timing, however, was significant, as Algorithm 3.4 was faster by a factor of roughly 33.

An example of the finite horizon MDPs that are constructed during solution of Example 3.1 are presented in Figure 3·4. The four MDPs that are constructed to solve this problem contain a total of 58 states in them, which is about a quarter of the number of states as the MDP constructed to solve the same problem with the

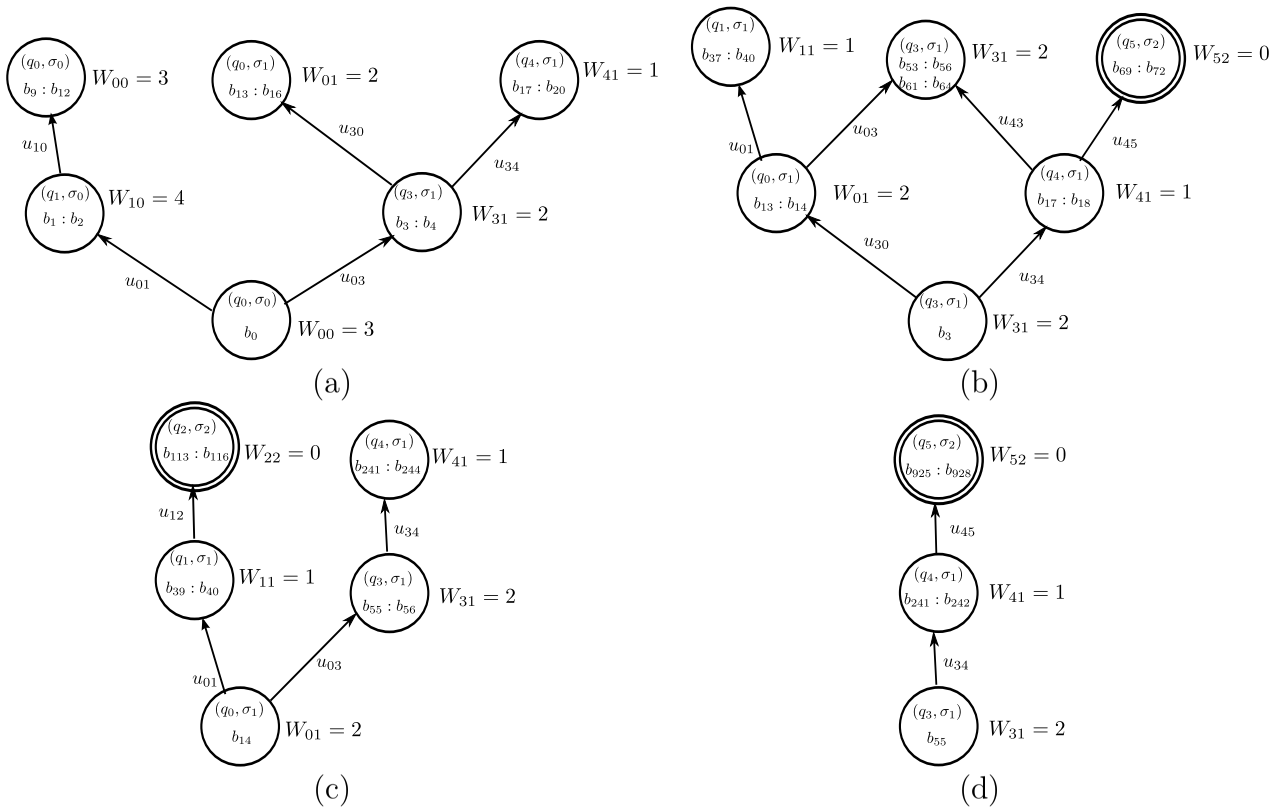


Figure 3·4: A sequence of MDPs constructed while executing Algorithm 3.4 with total horizon $\ell = 5$, planning horizon $m = 2$, and action horizon $n = 1$. Each state (q_i, σ_j, b) in the MDPs are annotated with the distance to accepting of the associated automaton state, $W_{ij} = W((q_i, \sigma_j))$ (a) The initial MDP constructed at time $k = 0$ with root (q_0, σ_0, b_0), depth $m = 2$, and time budget $\ell - k = 5$. (b) MDP constructed at time $k = 1$ after action u_{03} was taken and observation $y^1 = 0$ was seen in state q_3. The remaining budget is $\ell - k = 4$. (c) MDP at time $k = 2$ after action u_{30} was taken and observation $y^2 = 1$ was seen. The remaining budget is $\ell - k = 3$. (d) MDP at time $k = 3$ after action u_{03} was taken and observation $y^3 = 0$ was seen. The remaining time budget is $\ell - k = 2 = m$, which means that no more MDPs will be constructed (Algorithm 3.4, lines 7-8), and the optimal policy constructed on this MDP will be executed for the next two time periods. Note that only one choice can be made at this point.

Method	$H(\bar{b^t})$	$\sigma_{H(b^t)}$	$\bar{m(t)}$	$\sigma_{m(t)}$	$\bar{T}$	σ_T
Exact DP	3.50	0.49	1.70	0.62	28.71	0
Rec Hor DP	3.58	0.32	1.71	0.57	0.85	0.24

Table 3.1: Results from the case study simulation using 500 Monte Carlo simulations. Exact DP refers to Algorithm 3.1 and Rec Hor DP refers to Algorithm 3.4. Sample means denoted with bars and standard deviations noted as σ with subscripts.

same horizon using Algorithm 1 (Figure 3·3).

We also used this case study to evaluate empirically the effect of the planning horizon m and action horizon n on Algorithm 3.4. We varied the pairs (m, n) over the range $\{(m, n) \in \{1, \ldots, 7\}^2 | n \leq m\}$ and performed 100 Monte Carlo trials with each pair. The effect of varying m and n on average entropy and average mistake rates are shown in Figure 3·5(a) and (b),respectively.

The entropy and mistake rates both decrease overall as m grows larger and n grows smaller. These decreases, however, are not monotonic. We hypothesize that this non-monotonicity is not a sampling artifact. Rather, this relationship is likely heavily influenced by the transition probability of Env, the topology of the environment, and the specification ϕ. Each decision the receding horizon algorithm makes "prunes" a part of the full model MDP that is reachable. Because this pruning process occurs with incomplete information (i.e. receding horizon approaches are somewhat myopic), increasing the planning horizon may indeed lead to pruning more desirable solutions than having a shorter planning horizon.

Experimental Scenario

We complement the theoretical results and simulation of the running example with a larger data-driven simulation and an experimental case study. The scenario is as shown below. We control a pursuing aerial robot R_p that is tasked with localizing

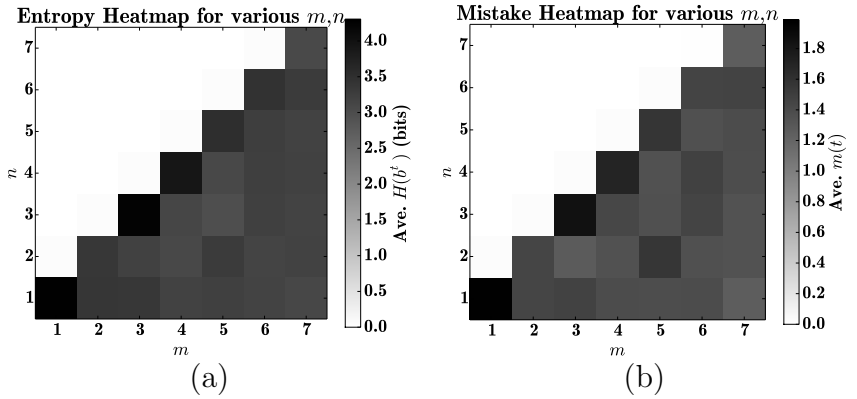


Figure 3·5: Heatmaps of (a) the average terminal entropy $H(b^t)$ and (b) the average mistake rates $m(t)$ for 100 Monte Carlo trials at each pair of (m, n) values.

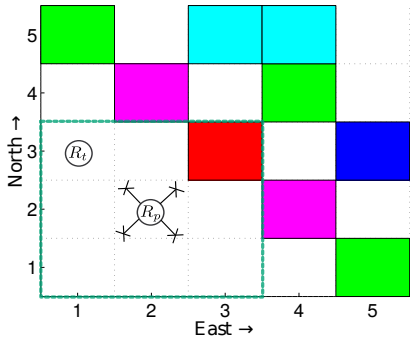
a target R_t on the ground (Figure 3·6a). The target R_t moves to an adjacent cell with probability p_{move}. R_p can move north, south, east, or west (N, S, E, W). R_p observes R_t via a downward facing camera. The field of view of the camera is one cell beyond R_p's position and denoted by the green outline in Figure 3·6(a). R_p's motion is constrained by the scLTL formula

$$\begin{aligned}\phi_{exp} = \quad & \diamondsuit\, p_1 \wedge \diamondsuit\, p_3 \wedge (\neg p_3 U p_1) \wedge (\neg p_2 U p_3) \wedge \\ & ((\diamondsuit\, p_4) \Rightarrow (\diamondsuit\, p_5 \wedge (\neg p_3 \mathcal{U} p_5)).\end{aligned} \tag{3.14}$$

The specification ϕ_{exp} can be translated to plain English as "Visit a magenta region (p_1) before visiting a green region (p_3). Always avoid the red region (p_2). If a blue region (p_4) is visited, then also visit a cyan region (p_5) before visiting a green region."

Data-Based Simulation

In this section, we performed a series of simulations of the scenario shown in Figure 3·6. We built the camera measurement model, i.e. the likelihood function h, from data collected from a real downward-facing camera on a quadrotor platform. We



(a)

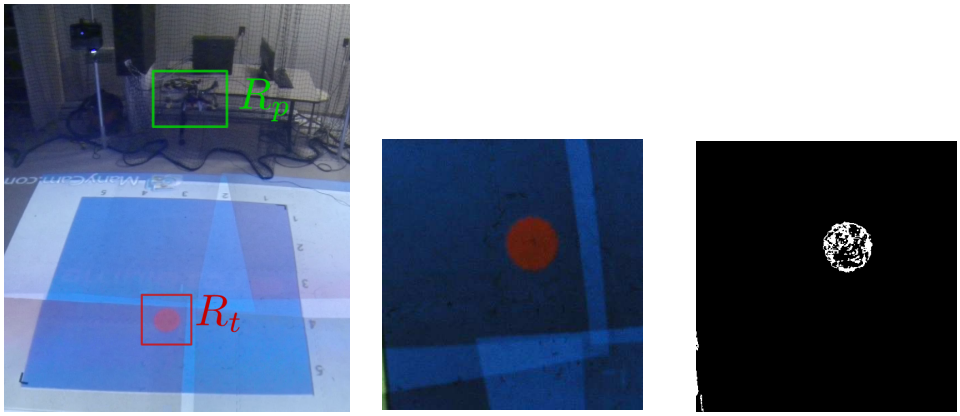
Prop.	color
p_1	magenta
p_2	red
p_3	green
p_4	blue
p_5	cyan

(b)

Figure 3·6: (a) Labeled, partitioned space in which the target robot (R_t) and pursuing robot (R_p) are operating. (b) Table showing the mapping between the color on the grid in (a) and the propositions used in specification (3.14).

then used this model in simulation to generate a large number of trials for statistical analysis.

The ground environment is simulated as a purple square projected via a series of short-throw projectors on the ground. The target R_t is simulated as a red circle on this purple background. We ran a set of experiments in which the position of R_t and the position of R_p were varied and a set of experiments in which R_t was not in the view of R_p. The quadrotor R_p was controlled over a wireless network and localized via an OptiTrack system. The experimental testbed used to collect the data is shown in Figure 3·7(a). For each configuration of R_p and R_t, we recorded 25 pictures from the downward-facing camera. A red color detection algorithm was run on each of the collected images as demonstrated in Figure 3·7. We hand-selected one of the images in which R_t was visible in R_p's field of view. From this image, we recorded the average and standard deviation of the red-green-blue (RGB) value in the area of R_t. Then, for a given image, the RGB value of each pixel was compared to the calibration data. We created a binary image where a pixel i is denoted as white if $C_i \in [\bar{C} - \eta sd_C, \bar{C} + \eta sd_C]$ where $C \in \{R, G, B\}$ is a color value and $\bar{C}, sd_C$ are the mean and standard deviation



(a) (b) (c)

Figure 3·7: (a) A picture of the experimental setup used to construct the measurement likelihood model. (b) Example image taken by the quadrotor's camera. (c) The binary image resulting from running the red detection algorithm on the picture from part (c).

of the calibration color data, respectively. Then, the binary image was split into four quadrants. The algorithm then returned a value $Y \in \{NP, NW, NE, SW, SE\}$ such that $Y = NP$ if the proportion of white pixels in any quadrant was than a value ν, and $Y = NW$, $Y = NE$, $Y = SW$, or $Y = SE$ if at least one quadrant has a proportion of white pixels greater than ν and the northwest, northeast, southwest, or southeast quadrants, respectively, had the highest proportion of white pixels. The frequency of the algorithm output for each set of images with fixed position of R_p and R_t were recorded and used to build the likelihood function h.

Note that the field of view of R_p actually encompasses nine possible locations of R_t, so the measurement model represents a down-sampling. This means that in order to localize R_t well, a single image in which R_t is in the field of view is not sufficient, so R_p has to "follow" R_t in order to gather enough data to localize R_t.

In our case, we set $\eta = 0.6$ and $\nu = 0.01$.

We used the noisy camera model to generate 100 trials each for when R_p moved according to the following five policies

1. Applying Algorithm 3.1 with $\ell = 6$,
2. Applying Algorithm 3.4 with $\ell = 6$,
3. A random walk with $\ell = 6$,
4. Applying Algorithm 3.4 with $\ell = 20$,
5. A random walk with $\ell = 20$.

The random walk is constrained to satisfy the scLTL specification, i.e., at each time i, the action a^i is chosen uniformly from the set of actions that $(\chi^i, a, \chi^{i+1}) \in Trans_{\mathcal{P}}$ and $W(\chi^{i+1}) \leq \ell - i - 1$. Algorithm 3.4 was executed with lookahead horizon $m = 3$ and action horizon $n = 1$. The histograms of the terminal entropy of each trial are shown in Figure 3·8. As we can see from the results when $\ell = 6$, both Algorithms 3.1 and 3.4 outperform the random walk. A two-sample t-test finds a difference between the performance of the random walk and Algorithm 3.1 with p-value with p-values less than 0.001. Interestingly, the difference between the performance of Algorithms 3.1 and 3.4 was not statistically significant, indicating that although 3.1 is guaranteed to outperform Algorithm 3.4, in practice this performance gap may be quite small for some applications. When $\ell = 20$, the difference between the random walk and Algorithm 3.4 is statistically significant with p-value less than 0.001. This indicates that when Algorithm 3.1 cannot be applied, the receding horizon implementation given here is also useful for gathering information.

The timing information for Algorithms 3.1 and 3.4 are summarized in Table 3.2. The simulations were performed on a machine with a 2.1 GHz processor with 7.4 GB RAM. Parallelization was not used. Algorithm 3.1 is a one-time calculation and hence has 0 standard deviation. The random walk algorithms took an average of less than 1s per trial. Note that Algorithm 3.1 requires nearly 100 times as much time (on

Method	ℓ	T (s)	σ_T (s)
Algorithm 3.1	6	727.161302	0
Algorithm 3.4	6	6.7681	1.1809
Algorithm 3.4	20	37.0905	14.0494

Table 3.2: Mean $\bar{T}$ and standard deviation σ_T of computation time required for Algorithms 3.1 and 3.4.

average) as 3.4 for only a small gain in performance. Meanwhile, when we increase the budget from 6 to 20, a 233 % increase, the average computational time required for Algorithm 3.4 increases only by 446 %. This relatively modest increase indicates further that Algorithm 3.4 may be more appropriate for practical applications.

Figure 3·9 shows the average value with one standard deviation error bars for the estimation error, given as

$$err = 1 - b^t(s^t). \tag{3.15}$$

That is, the estimation error is the belief that R_t is not in its true location in the environment. The performance advantage in terms of error of our algorithms over the random walk are all statistically significant with p-values less than 0.001. The difference in performance between Algorithms 3.1 and 3.4 was not statistically significant.

Experimental Validation

To help validate the findings of our simulation case study, we performed a set of experiments in the same testbed used to gather the camera data. During the experimental trials, whenever R_p moves to a new location in the environment, it hovers and takes a picture with its camera. The image is streamed to a computer and the red detection algorithm is applied. The output is then used to find the next action

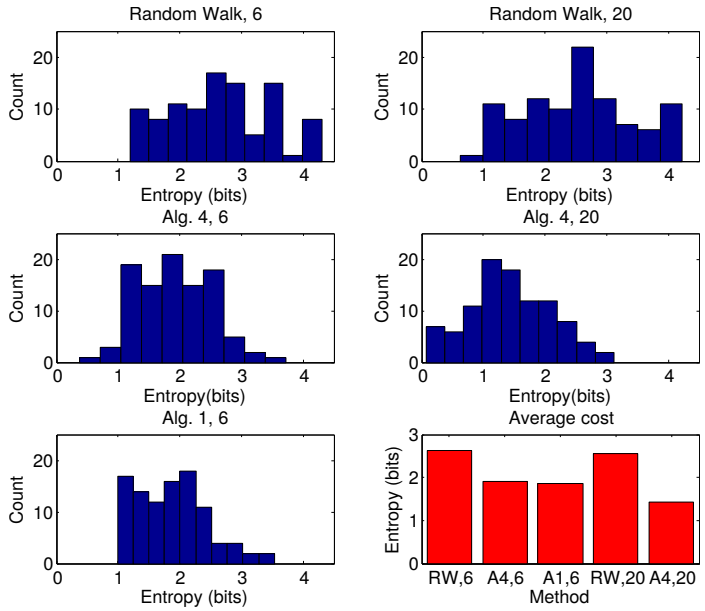


Figure 3·8: Histograms of simulation results and a bar graph showing the average cost of each method (lower right). Each plot is labeled with the corresponding policy used to control R_p (Algorithm 3.1, Algorithm 3.4, or random walk) and the budget given to the policy (6 or 20).

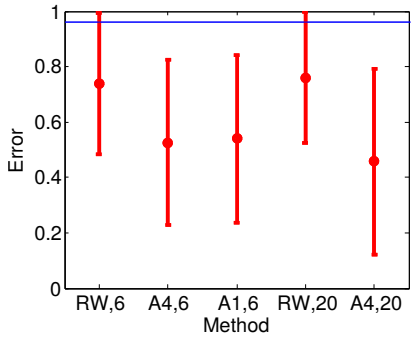


Figure 3·9: Plot of the average error (3.15) with error bars indicating the standard deviation. The blue line indicates the error level of the uniform distribution.

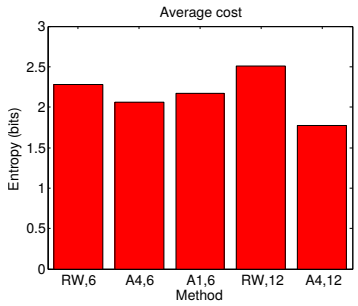


Figure 3·10: Average costs for the 10 trials of each experiment.

R_p takes. In this case, we performed 10 experiments each in which R_p was moving according to the five policies used in simulation. Due to battery power constraints, the budget ℓ was shortened from 20 actions to 12 actions for the two long-term tests. Partial trials that failed due to mechanical or battery failure were ignored. The average costs are summarized in Figure 3·10. The three costs with budget $\ell = 6$ are comparable, though both Algorithms 3.1 and 3.4 outperformed the random walk. When $\ell = 12$, a two-sample t-test concludes that the difference between the random walk and Algorithm 3.4 for the experiment was statistically significant with p-value 0.02135.

Note that the average costs gathered in the experiment are slightly larger than for the simulations. This is because the camera model we constructed is not an exact match for the on-line performance of the camera, as levels of background illumination may change over time. However, the difference between the random walk and Algorithm 3.1 with $\ell = 12$ indicates that even when the sensor model is not an exact match to the on-line model, our optimization procedures still result in better estimation than randomized search.

3.3 Infinite-Horizon Informative Planning

In this section, we address the problem of persistent monitoring under LTL constraints. This describes many real-world applications in which human decision-makers require real-time information about large environments, such as forest rangers monitoring wildfires or traffic engineers monitoring congestion.

We formally state the problem under consideration in Section 3.3.1. In Section 3.3.2, we extend the receding-horizon planner we developed in (Jones et al., 2015b) (Section 3.2.3) to guarantee satisfaction of LTL constraints. In addition, to avoid myopia inherent in receding horizon implementations, we develop a high-level simulation-based planner that selects inputs to the receding horizon algorithm that attempt to maximize the MIR over the infinite horizon. An implementation of our procedure is applied to a simulation of a target tracking case study in Section 3.3.3.

3.3.1 Problem Definition

Our goal is to plan an infinite horizon trajectory $q^{0:\infty}$ for *Robot* that maximizes the quality of the estimate pmf over time while satisfying a linear temporal logic specification. In (Jones et al., 2013b; Jones et al., 2015b), we used the expected conditional entropy $E_{Y^{1:t}}[H(b^t)]$ to quantify the expected quality of the estimate that would result from the robot traveling along the finite path $q^{0:t}$. Here, we use a related quantity called the mutual information rate (Cover and Thomas, 2006; Shannon, 1948)

$$MIR = \lim_{t\to\infty} \frac{I(s^{0:t}; Y^{0:t})}{t}. \tag{3.16}$$

MIR is the average rate of information gain about the state of Env when a new measurement is taken. We wish to maximize this quantity, as we want to increase the rate at which measurements help to identify the state of Env. Maximizing (3.16)

over the set of actions *Robot* can take is equivalent to minimizing the *entropy rate*

$$ER = \lim_{t\to\infty} \frac{H(b^t)}{t}. \tag{3.17}$$

Due to the "infinitely often" acceptance condition of LTL formulae, any trajectory that satisfies an LTL formula can be segmented into a finite length prefix path and an infinite sequence of finite length suffix cycles, which are defined as follows.

Definition 3.2 (Prefix Path and Suffix Cycle)**.** Let $q^{0:\infty}$ be an infinite horizon trajectory over a transition system TS and ϕ be an LTL formula over the properties AP in TS. Let $\mathcal{P} = TS \times \mathcal{B}_\phi$ and let $\chi^{0:\infty}$ be the infinite run over $\mathcal{P}$ induced by $q^{0:\infty}$. Let l_n be the nth time at which $\chi^{0:\infty}$ intersects F_P. The *prefix path* is the finite sequence $q^{0:l_1}$. A *suffix cycle* is a finite sequence of states $q^{l_n+1:l_{n+1}}$.

We wish to minimize the average entropy rate per cycle, given as

$$AERPC = \lim_{n\to\infty} \frac{\sum_{i=0}^{n} \frac{H(b^{l_n}) - H(b^{l_n-1})}{l_n - l_{n-1}}}{n} \tag{3.18}$$

This formulation is similar to the average cost-per-stage problem (Ding et al., 2010; Bertsekas, 2000). With this new objective, we define the constrained persistent monitoring problem as

Problem 3.2. Consider an agent that is estimating the state of the environment. Solve

$$\begin{gathered} \min_{a^0\ldots\in Act^\infty} E_{Y^{0:l_\infty}}[AERPC] \\ \text{subject to} \\ \phi \\ l_{n+1} - l_n \leq \ell_{max} \forall n, \end{gathered} \tag{3.19}$$

where ϕ is an LTL formula and ℓ_{max} is a per-cycle budget.

The budget ℓ_{max} describes energy constraints on the agent's motion, e.g. the

maximum amount of time that the agent can be active between recharging. We map Problem 3.2 to the constrained stochastic optimal control problem over the full model MDP (Section 3.2.1)

$$\begin{gathered}\min_{a^0\ldots\in Act^\infty} E_{Y^{0:l_\infty}}[AERPC] \\ \text{subject to} \\ l_{n+1} - l_n \leq \ell_{max} \forall n \in \mathbb{N}, \\ a^i \in Act((\chi^i, b^i)) \ \forall i \in \mathbb{N} \\ (\chi^{i+1}, b^{i+1}) \sim P_{tot}(\cdot, a^i, (\chi^i, b^i)) \ \forall i \in \mathbb{N}.\end{gathered} \tag{3.20}$$

3.3.2 Algorithms

We propose a hierarchal dynamic programming-based method to solve this problem. We use a low-level receding-horizon algorithm (defined in Section 3.3.2) to choose actions that locally improve the entropy rate and drive the agent to an accepting state in $\mathcal{P}$. Whenever the agent completes a suffix cycle, the budget of actions given to the receding horizon algorithm is selected according to a high-level policy that is calculated off-line via value iteration (Bertsekas, 2000) to optimize the infinite-horizon AERPC (Section 3.3.2).

Optimization of a single cycle

In (Jones et al., 2015b), we presented a receding horizon algorithm that locally minimized entropy and was guaranteed to satisfy the given scLTL constraint within the budget. Algorithm 3.5 extends that procedure to handle LTL constraints.

At the kth time-step after the invocation of Algorithm 3.5, the procedure PossibleStates constructs m sets $\Sigma_p[i] \subseteq \Sigma_{\mathcal{P}}$ where Σ_p contains all states $\chi' \in \partial N(\chi[k], i)$ such that $W(\chi') \leq \ell - i - k$, that is, all states reachable from $\chi[k]$ in i actions and from which an accepting state can be reached under the remaining budget. From the sets Σ_p, the procedure ConstructMDP constructs a subset of the full-model MDP

Algorithm 3.5 Receding horizon planner for completing a single suffix cycle under LTL constraints.

function RHP$((\chi, b), \ell, m, n)$
$k = 0$; $\chi[k] = \chi$;
while $\chi[k] \notin F_{\mathcal{P}}$ **do**
  $\Sigma_p :=$ PossibleStates($\chi[k]$,m,$\ell - k$)
  $(X_p$,P_{tot},$Act_p) :=$ ConstructMDP($\chi[k]$,b,Σ_p)
  $\mu_l :=$BellmanIteration(X_p,P_{tot},Act_p)
  if $k \geq \ell - m$ **then**
    $n := \ell - k$
  for $i := 1$ to n **do**
    $(\chi[k+1], b) :=$ result from applying $\mu(i, (\chi, b))$
    $k++$
return $(\chi[k], b, k)$

such that at the ith level, only pairs (χ, b) such that $\chi \in \Sigma_p[i]$ appear. Next, the algorithm BellmanIteration performs standard Bellman iteration over the constructed MDP where the terminal cost is given as

$$\frac{H(b[m]) - H(b[0])}{m}, \tag{3.21}$$

where $b[m]$ is the terminal state and $b[0]$ was the initial belief state input to Algorithm 3.5. After the optimal policy μ_l has been calculated, it is enacted for n time steps. At this point, a new MDP is constructed and a new policy calculated. This process continues until an accepting state is reached.

Applying Algorithm 3.5 infinitely often provably satisfies the given specification.

Theorem 3.3. *Let*

$$\ell' = \max_{\chi \in F_{\mathcal{P}}} \min_{\chi' \in \partial N(\chi, 1)} W(\chi') + 1.$$

If $\ell' \leq \ell_{max}$, then sequentially applying Algorithm 3.5 infinitely often with budget at least ℓ' is guaranteed to drive the robot to satisfy the given LTL specification.

Proof. Algorithm 3.5 is always applied either at the initial state (χ_0, b^0) or at an

accepting state (χ, b) such that $\chi \in F_{\mathcal{P}}$. Since we constrain ℓ to be at least ℓ', it is guaranteed that $W(\chi) \leq \ell$. k time steps after the algorithm is applied, the system will be in a state (χ', b') such that $W(\chi') \leq \ell - k$. This means that if $k = \ell$, $W(\chi') = 0$, i.e. $\chi' \in F_{\mathcal{P}}$. So, applying Algorithm 3.5 one time is guaranteed to drive the system to an accepting state. Applying Algorithm 3.5 infinitely often guarantees that the system will be in an accepting state infinitely often, thus satisfying the Buchi acceptance condition and therefore satisfying the given LTL specification. □

Further, a single application of Algorithm 3.5 is tractable.

Lemma 3.3. *The time complexity of Algorithm 3.5 with budget ℓ, planning horizon m, and action horizon n is $O(|Act|^m |R_Y|^m |S| \lceil \frac{\ell}{n} \rceil)$.*

Choosing cycle budgets

Although applying Algorithm 3.5 infinitely often with a fixed budget ℓ is guaranteed to satisfy the LTL specification, there is no guarantee that the local information gathering performs well over an infinite time horizon. We propose to pair the local information gathering with long-term planning by finding a policy $\mu_g : F_{\mathcal{P}} \cup \{\chi_0\} \times B \to \mathbb{N}$ that maps an initial or accepting state in the automaton and a belief state, e.g. the configuration of the robot at the end of a cycle, to the budget ℓ that should be given to the receding horizon planner in the next cycle. The hierarchal structure of the algorithm is illustrated in Figure 3·11.

It may seem that the longer the budget ℓ handed to the receding horizon planner, the better we expect the performance to be. As ℓ increases, on average more states are included in the MDPs constructed by Algorithm 3.5. However, the agent can take an action that locally performs better but which may cause the long-term performance to deviate far from optimal behavior. If this action were not present, the deviation

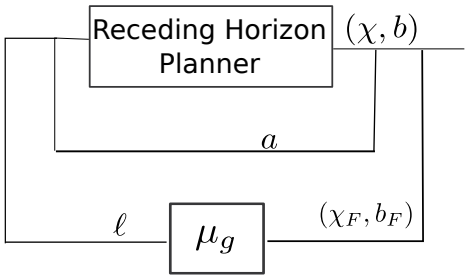


Figure 3·11: A block diagram illustrating the hierarchal dynamic programming algorithm developed in this section. The receding horizon planner applies Algorithm 3.5 to determine the next action to take based on the current automaton state and belief state pair (χ, b) and the budget of actions ℓ between satisfactions. When the system reaches an accepting configuration, denoted (χ_F, b_F), the policy μ_g is called to determine the budget ℓ that should be used in the next round of receding horizon planning.

would not occur. We address this by characterizing the statistical performance of the receding horizon algorithm for different budgets and use this information to construct the optimal policy μ_g. The policy construction is performed off-line before the agent is deployed.

Algorithm 3.6 details how μ_g is calculated. We use simulation to characterize the per-cycle performance of the receding horizon algorithm. We consider a finite set of states $T \subset F_{\mathcal{P}} \times B$ in our optimization. Algorithm 3.6 begins by performing j_{max} simulations of Algorithm 3.5 from the initial state (χ^0, b^0) for each value of ℓ from $\min_{\chi'' \in \partial N(\chi^0, 1)} W(\chi'') + 1$ to ℓ_{max}. The result of each simulation is a state (χ', b') that is reachable from the initial state. Note that in our algorithm, we group similar belief states together. If two states are ϵ close in the 1-norm, we consider them identical. This is to help mitigate the state explosion problem and to prevent unnecessary calculation of similar quantities. The cost of going from (χ, b) to (χ', b') under budget ℓ, denoted $g((\chi, b), \ell, (\chi', b')) = \frac{H(b') - H(b)}{\ell}$, is calculated and recorded. Similarly, the frequency of transitioning from state (χ, b) to (χ', b'), under budget ℓ, denoted $P((\chi, b), \ell, (\chi', b'))$, is calculated from the set of simulations and recorded. The reachable state (χ', b') is then added to the set the set of states still to be checked C. After

this set of simulations is completed, the states of C are iterated through and more simulations occur.

T, P, and g are populated until $|T| \geq N$ and all of the accepting states in $F_{\mathcal{P}}$ have been visited or there are no states left to be checked (due to all of the states found being ϵ-close to members of T). At this point, we invoke a procedure to make the MDP described by P,g, and T recurrent. That is, for every state $(\chi, b) \in T$ for which P is undefined, we apply Algorithm 3.5 for each possible value of ℓ. Then, we replace the resulting state (χ', b') with (χ_c, b_c) the closest state already enumerated. The resulting costs $g((\chi, b), \ell, (\chi_c, b_c))$ and transition probabilities $P((\chi, b), \ell, (\chi_c, b_c))$ are recorded. In short, this procedure ensures that every state in T is connected via P to another state in T. Finally, Algorithm 3.6 uses value iteration to find the optimal policy

$$\begin{aligned} \mu_g(\chi, b) = \quad & \arg\min_{\ell} \textstyle\sum_{(\chi', b') \in T} P((\chi, b), \ell(\chi', b')) \\ & [g((\chi, b), \ell(\chi', b') + J^{\infty}((\chi', b'))], \end{aligned} \tag{3.22}$$

for all $(\chi, b) \in T$ where $J^{\infty} : T \to \mathbb{R}$ is the infinite-horizon cost-to-go function. (Bertsekas, 2000) Note that this optimization does not directly minimize the average entropy rate per cycle. Rather, it minimizes the sum of the expected entropy rates per cycle. However, since we are minimizing the entropy rate on a cycle-by-cycle basis, the average rate will also be minimized.

Algorithm 3.7 summarizes our approach. We calculate the optimal budget policy off-line. Then, we use the policy on-line when applying Algorithm 3.5 infinitely often. If a state is reached that is not in the pre-calculated set T, we apply the mapping μ_g to the closest state contained in T.

Algorithm 3.6 Constructs the policy mapping current state to optimal budget.

function OptimalBudget(χ_0,b^0,ℓ_{max},m,n)
$T = \{(\chi_0, b^0)\}$; $C = \{(\chi_0, b_0)\}$; $Acc = F_{\mathcal{P}}$
while $|T| \leq N$ **do**
  $(\chi, b) = C$.pop()
  for $\ell = \min_{\chi'' \in \partial Nin} W(\chi'') + 1$ to ℓ_{max} **do**
    for $j = 1$ to j_{max} **do**
      $(\chi', b', k) = \text{RHP}((\chi, b), \ell, m, n)$
      if $\exists (\chi', b'') \in T$ s.t. $||b'' - b'||_1 < \epsilon$ **then**
        $b' = b''$
      else
        T.add((χ', b')); C.add((χ', b'))
      $g((\chi, b), \ell, (\chi', b')) = \frac{H(b') - H(b)}{k}$
      $P((\chi, b), \ell, (\chi', b'))$ += $\frac{1}{j_{max}}$
      $Acc = Acc \setminus \{\chi'\}$
  if $C = \emptyset$ **then**
    Break;
  if $|T| = N$ and $Acc \neq \emptyset$ **then**
    $N = N + 1$
$(P, g, T) = \text{MakeRecurrent}(P, g, T)$
$\mu_g = \text{ValueIteration}(P, g, T)$
return μ_g, T

Algorithm 3.7 Solution to constrained persistent monitoring

function PersistentMonitor(χ_0,b^0,ℓ_{max},m,n)
$\mu_g, T = \text{OptimalBudget}(\chi_0, b^0, \ell_{max}, m, n)$;
$\chi = \chi_0$; $b = b^0$;
while TRUE **do**
  if $(\chi, b) \notin T$ **then**
    $b = \arg\min_{b' \in T} ||b - b'||_1$
  $(\chi, b, k) =$RHP$((\chi, b), \mu_g((\chi, b)), m, n)$

3.3.3 Case Studies

We implemented Algorithm 3.7 in software and applied it to a simulation of the scenario described in Example 3.2.

Example 3.2. Consider a robot R_p operating in a grid environment as shown in Figure 3·12(a) to track a target robot R_t. In this case, $Q = \{1, \ldots 4\}^2 \times \{N, S, E, W\}$ (north, south, east, west) where a state $q = (i, j, dir)$ means that R_p is in grid cell (i, j) and facing direction dir. The set of actions is $Act = \{straight, CW, CCW\}$ which mean go straight, rotate 90° clockwise, and rotate 90° counterclockwise, respectively. $AP = \{\pi_{recharge}(\text{green}), \pi_{data}(\text{magenta}), \pi_{alarm}(\text{cyan}), \pi_{reset}(\text{blue}), \pi_{obs}(\text{red})\}$. A subset of the transition system is shown in Figure 3·12(b).

In the above scenario, $S = \{1, \ldots, 4\}^2$ is the set of possible locations of R_T. s^0 is chosen randomly. Env can transition to an adjacent state with probability p_{move}. R_p has a noisy camera that can produce measurements in $R_Y = \{0, l, c, r\}$ which correspond to R_t not being observed and R_t being observed in the left, center, or right portion of R_p's field of view. Part of the measurement likelihood function is summarized in Figure 3·12(c). The constraints on R_p can be given as the LTL formula

$$\begin{aligned} \phi = \quad & \square \diamondsuit \pi_{recharge} \wedge \square \diamondsuit \pi_{data} \wedge \square (\neg \pi_{obs}) \\ & \wedge \square (\pi_{alarm} \Rightarrow (\neg \pi_{recharge} \mathcal{U} \pi_{reset})), \end{aligned} \tag{3.23}$$

which in plain English is "Visit recharging and data upload stations infinitely often while avoiding obstacles. If an alarm is triggered, visit the shutdown switch before visiting the recharging station." Figure 3·12(c) shows a prefix cycle and a suffix cycle for ϕ with blue and orange arrows, respectively. □

The probability that the tracked target R_t moves to an adjacent cell is $p_{move} = 0.15$. The optimal policy μ_g was computed with parameters $\epsilon = 0.01$, $N = 1500$,

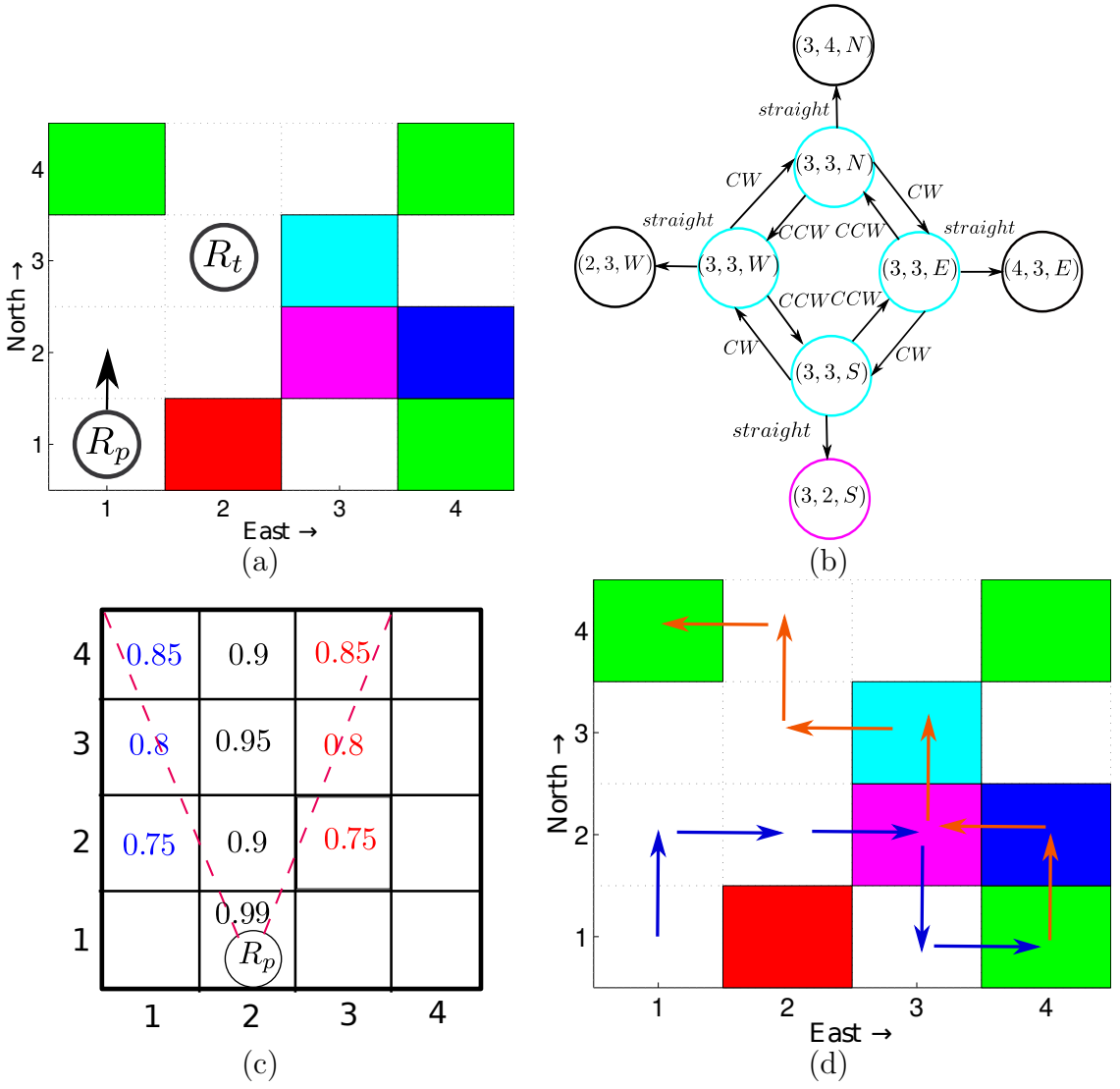


Figure 3·12: (a) Environment used in Example 3.2. (b) Part of transition system corresponding to (a). (c) Part of the measurement likelihood function. Numbers in blue, black, and red text indicate the probability of l, c, or r being measured by R_p if R_t is in the indicated cell. (d) Example of a prefix (blue) and a suffix cycle (orange) from formula 3.23.

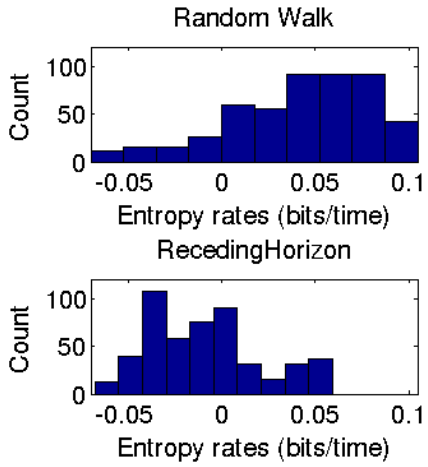


Figure 3·13: (a) random walk policy (b) Algorithm 3.7.

$j_{max} = 100$, $\ell_{max} = 15$, $m = 2$, and $n = 1$. The computation required approximately 6 hrs. of processor time. We performed 500 Monte Carlo trials of the system in which the system was simulated until 5 suffix cycles were completed, i.e. until R_p had visited a recharging station and a data upload station 5 times. This was compared to 500 "random walk" simulations in which at every time $k+l_n$, the agent selected its action uniformly at random from the set $\{a \in Act | (q^{k+l_n}, a, q') \in Trans, W(q') \leq \ell_{max} - k\}$. This random walk policy is also guaranteed to satisfy the constraints on the system. The results from the simulations are summarized in the histograms in Figure 3·13. The random walk policy resulted in an average entropy rate of 0.0420 bits/action, while the average entropy rate when using Algorithm 3.7 was -0.0090 bits/action. A two-sample t-test confirmed that this difference in means is statistically significant with p-value less than 10^{-95}. Further, 3.5 % of the random walk trials resulted in negative entropy rates while 64.8% of the trials with Algorithm 3.7 had negative rates, i.e. on average gain information about the location of R_t with each action taken. This indicates that in addition to having better mean performance, Algorithm 3.7 has better performance more often.

Chapter 4

Temporal Logic Tasks under Uncertainty

In this chapter, we define Distribution Temporal Logic (DTL), a new kind of TL for specifying tasks for stochastic systems with partial state information. DTL leverages rich information in the belief space that is currently unexploited. We originally defined DTL in (Jones et al., 2013a). The logic is well-suited to problems in which state uncertainty is significant and unavoidable, and the state is estimated on-line. Many such systems arise in robotics applications, where a robot may be uncertain of, for example, its own position in its environment, the location of objects in its environment, or the classification of objects (e.g. 'target' or 'obstacle').

We define the model under study, partially observable Markov decision processes (POMDPs) in Section 4.1. Next, we present a motivating hypothesis testing scenario that demonstrates that the paradigm of DTL is more expressive than existing temporal logics in Section 4.2. We then give a precise definition of scLDTL, a finite-horizon DTL, in Section 4.3. We present an algorithm for determining *ex post facto* with what probability a given sample path of a POMDP satisfies a given scLDTL specification in Section 4.4. Finally, we validate this monitoring algorithm by using it to compare the performance of two different control policies over a POMDP in Section 4.5.

4.1 Partially Observable Markov Decision Processes

POMDPs (Monahan, 1982; Kaelbling et al., 1998; Varakantham et al., 2006) are extensions of HMMs in which actions can be taken to affect the probabilistic evolution of the hidden states and the observation process. More formally,

Definition 4.1 (POMDP)**.** A POMDP is a tuple $POMDP = (S, b^0, P, Act, Obs, h)$ where S is a set of (hidden) states of the system, Act is a collection of actions, and $P : S \times Act \times S \to \mathbb{R}$ is a probabilistic transition relation such that if $POMDP$ is in a state s, taking the action a will drive the system to state s' with probability $P(s, a, s')$. After the hidden state evolves, the system generates an observation from the set Obs with probability $h(s, a, o) = Pr[o \text{ seen } |a \text{ taken, } POMDP \text{ in state } s]$. The system maintains a belief state b^t of the current state of $POMDP$, where $b^t(s) = Pr[POMDP \text{ in state } s \text{ at time } t|a^{0:t-1} \text{ taken, } o^{1:t} \text{ seen}]$, via sequential application of the recursive Bayes filter (3.2) initialized with the prior distribution b^0. □

The belief state also has a geometric exploitation that we exploit in our monitoring algorithm in Section 4.4.

Definition 4.2 (Belief Simplex)**.** Given a POMDP $POMDP = (S, b^0, P, Act, Obs, h)$, its belief state has an equivalent vector representation in the *belief simplex* Δ^{n-1} where

$$\Delta^m = \{[z_1 \dots z_m] | \sum_{i=1}^{m} z_i \leq 1, z_i \geq 0 \ \forall i \in [1, m]\} \tag{4.1}$$

is the m- dimensional *belief simplex* . The ith element of the vector is equal to $b(s_i)$. The value of $b(s_n)$ can be recovered uniquely from the probability vector via $b(s_n) = 1 - \sum_{i=1}^{n-1} b(s_i)$. □

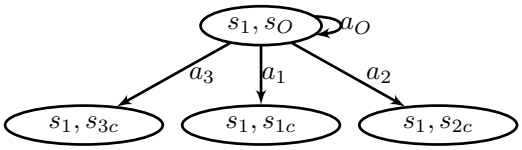


Figure 4·1: A representation of the hidden state dynamics of the multiple hypothesis testing POMDP given that the true source is s_1. The full state dynamics is given by three separate graphs of the same form where the first element s_i in the state tuple indicates the true source of the observation sequence.

4.2 Motivating Example: Hypothesis Testing

In this section, we use a simple multiple hypothesis testing example to motivate the introduction of the logic scLDTL described in Section 4.3. Consider an experiment in which one of three coins, each with different expected frequency of heads, is flipped repeatedly. This is an example of a Hidden Markov Model (HMM). The hidden states of the system are $S_h = \{s_1, s_2, s_3\}$, where s_i is a coin with heads frequency p_i. The set of observations is $Obs = \{o_1, o_2\}$ where o_1 is heads and o_2 is tails.

Further, consider a deciding agent that at each time step can either make an observation from the HMM or choose a hypothesis in S_h. Let $S = S_h \times S_d$, where $S_d = \{s_{1c}, s_{2c}, s_{3c}, s_O\}$ is the state space of the deciding agent. s_O means that the HMM is being observed and s_{ic} means that the hidden state s_i is chosen as the most likely hypothesis. The process is illustrated in Figure 4·1. Combining the observation model from the HMM with the state dynamics described by Figure 4·1 gives a POMDP $MHT = (S, b^0, P, \{a_O, a_1, a_2, a_3\}, \{o_1, o_2\}, h)$ where P and h are given by

$$\begin{array}{ll} P([s_i, s_0], a_0, [s_i, s_0]) = 1, & \\ P([s_i, s_0], a_j, [s_i, s_{jc}]) = 1 & \forall i, j \in \{1, 2, 3\} \\ P(s, a, s') = 0, & \text{otherwise} \end{array} \tag{4.2a}$$

$$\begin{array}{ll} h([s_i, s_O], a_O, o_1) = p_i, & h([s_i, s_O], a_O, o_2) = 1 - p_i \\ h(s, a, o) = 0, & \text{otherwise} \end{array} \tag{4.2b}$$

Consider the problem in which we are given an infinite number of observations from MHT, but must estimate the state of the system in finite time. One solution method is to prescribe a threshold on the entropy of the belief state and terminate observation and select the most likely hypothesis when it is reached. In plain English, this is "When the entropy of the belief state is below h, select the most likely hypothesis."

This can easily be described by the new Distribution Temporal Logic (DTL) we define in Section 4.3. As it will become clear later, this predicate logic is defined over two types of predicates: belief predicates and state predicates. "When the entropy of the belief state is below h" is equivalent to the belief predicate $H(b) - h < 0$, which can be written in short as $(H(b) - h)$ where $H(\cdot)$ denotes entropy. "The most likely hypothesis" is equivalent to s_i such that $b([s_i, s_O]) > b([s, s_O])\ \forall s \in S_h \setminus \{s_i\}$. Each comparison between components of b is a belief predicate. The selection of hypothesis s_i means the state is in the set $\{[s_j, s_{ic}]\}_{j\in\{1,2,3\}}$, and such sets will be referred to as state predicates. As it will become clear in Section 4.3, the overall specification translates to the following DTL formula

$$\begin{array}{l}(H(b) - h) \Rightarrow \\ (\bigwedge_{s_i \in S_h}(\bigwedge_{s_j \in S_h \setminus \{s_i\}}(b([s_j, s_O]) - b([s_i, s_O])) \Rightarrow \\ \bigcirc\{[s_j, s_{ic}]\}_{j\in\{1,2,3\}}),\end{array} \tag{4.3}$$

where the temporal and logical operators have roughly the same semantics as scLTL (see Section 4.3, Definition 4.4).

Neither the threshold on entropy nor the selection of the most likely hypothesis can be formulated using POCTL*, the existing temporal logic for partially observable Markov chains (Zhang, 2004). POCTL* can describe some properties with respect to a belief state, namely whether the probability under the initial belief state of a

collection of sample paths of hidden states and observations occurring is greater than or less than some threshold, but this calculation is a linear function of the belief state. As entropy is a non-linear function of the belief state, entropy levels cannot be described in POCTL*.

The collection of sample paths that can be produced by the hidden state of the system are infinite repetitions of the s_i. The probability of sample path $s_i s_i \ldots$ under a belief state is $b([s_i, s_O])$. In POCTL* for this problem, we can only compare the probability under a belief state of a single hypothesis or a pair of hypotheses to a constant value: we cannot compare the estimated probabilities of hypotheses to each other. Therefore, we cannot use POCTL* to formulate the selection of the most likely hypothesis.

Since the problem we consider here is readily addressed with tools from optimal estimation and information theory (see e.g. (Cover and Thomas, 2006; Scharf and Demeure, 1991)), constructing a new TL to describe the solution strategy may seem unnecessary. However, even considering only belief predicates that describe measures of uncertainty allows the description of novel behaviors. We can specify low uncertainty levels as temporal goals. We can use uncertainty thresholds to trigger behavior consistent with the most likely state(s) of the POMDP.

4.3 Syntactically Co-Safe Linear Distribution Temporal Logic

Syntactically co-safe linear distribution temporal logic (scLDTL) describes co-safe temporal logic properties of probabilistic systems and is defined over two types of predicates: belief predicates of the type $f < 0$, with $f \in F_S : \{f : Dist(S) \to \mathbb{R}\}$ (denoted simply by f) where $Dist(S)$ is the set of all pmfs that can be defined over state space S and state predicates $s \in A$, with $A \in 2^S$ (denoted simply by A).

Formally, we have:

Definition 4.3 (scLDTL syntax)**.** An scLDTL formula over predicates over F_S and state sets is inductively defined as follows:

$$\phi := A|\neg A|f|\neg f|\phi \vee \phi|\phi \wedge \phi|\phi \mathcal{U} \phi|\bigcirc \phi|\ \diamondsuit\ \phi, \tag{4.4}$$

where $A \in 2^S$ is a set of states, $f \in F_S$ is a belief predicate, ϕ is an scLDTL formula, and $\neg$, $\vee$, $\wedge$, $\bigcirc$, $\mathcal{U}$, and $\diamondsuit$ are as described in Section 2. □

As scLDTL is defined over state and belief predicates, we construct a basic notion of satisfaction over pairs of hidden state sample paths and sequences of belief states, given by Definition 4.4.

Definition 4.4 (scLDTL satisfaction of sample path/belief state sequence pairs)**.** The semantics of scLDTL formulae is defined over words $w \in (S \times Dist(S))^*$. Denote the ith letter in w as (s^i, b^i) The satisfaction of a scLDTL formula at position i in w, denoted $(s^i, b^i) \models \phi$, is recursively defined as follows:

- $(s^i, b^i) \models A$ if $s^i \in A$,
- $(s^i, b^i) \models f$ if $f(b^i) < 0$,
- $(s^i, b^i) \models \neg A$ if $s^i \notin A$,
- $(s^i, b^i) \models \neg f$ if $f(b^i) \geq 0$,
- $(s^i, b^i) \models \phi_1 \wedge \phi_2$ if $(s^i, b^i) \models \phi_1$ and $(s^i, b^i) \models \phi_2$,
- $(s^i, b^i) \models \phi_1 \vee \phi_2$ if $(s^i, b^i) \models \phi_1$ or $(s^i, b^i) \models \phi_2$,
- $(s^i, b^i) \models \bigcirc \phi$ if $(s^{i+1}, b^{i+1}) \models \phi$,

- $(s^i, b^i) \models \phi_1 \mathcal{U} \phi_2$ if there exists $j \geq i$ such that $(s^j, b^j) \models \phi_2$ and for all $i \leq k < j$ $(s^k, b^k) \models \phi_1$,
- $(s^i, b^i) \models \ \diamondsuit \ \phi$ if there exists $j \geq i$ such that $(s^j, b^j) \models \phi$.

The word $w \models \phi$, iff $(s^0, b^0) \models \phi$. □

We also define a notion of probabilistic satisfaction with respect to an execution of a POMDP in Definition 4.5.

Definition 4.5 (scLDTL satisfaction with respect to a POMDP execution)**.** An execution of a POMDP (a sequence of belief states $b^{0:t}$, the sequence of actions taken $a^{0:t-1}$, and the sequence of observations seen $o^{1:t}$) *probabilistically satisfies* the scLDTL formula ϕ with probability $Pr[\{s^{0:t} \text{ such that } (s^0, b^0) \dots (s^t, b^t) \models \phi\} | b^{0:t}, a^{0:t-1}, o^{1:t}]$, denoted in shorthand as $Pr[\phi | b^{0:t}, a^{0:t-1}, o^{1:t}]$. □

The probability of a single sample path conditioned on a POMDP execution may be calculated via the process of recursive smoothing (Briers et al., 2010). Note that we define $Pr[\phi | b^{0:t}, a^{0:t-1}, o^{1:t}]$ with respect to finite-length sample paths. Although the semantics of scLDTL is defined over infinite words, it is known that any co-safe temporal logic formula can be checked for satisfaction in finite time (Latvala, 2003).

4.4 Monitoring POMDPs

Here we show algorithmically how to solve the following problem.

Problem 4.1 (scLDTL monitoring of POMDPs)**.** Evaluate with what probability a given finite-length execution of a POMDP $POMDP = (S, b^0, P, Act, Obs, h)$ satisfies an scLDTL formula ϕ over subsets of S and belief states over S. □

The solution to this problem could be used to evaluate the performance of a single execution of a POMDP or, as we show in Section 4.5, can be used to compare the performance of control policies. More importantly, the tools developed for this problem are potentially useful for developing synthesis procedures.

The evaluation proceeds in two stages. In the first stage, called feasibility checking, we check a necessary condition for the given execution to satisfy ϕ with $Pr[\phi|$ $b^{0:t}, a^{0:t-1}, o^{1:t}] > 0$. The second stage is probabilistic satisfaction checking, in which $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}]$ is calculated if feasibility checking has succeeded.

4.4.1 Feasibility checking

Algorithm 4.1 shows how to construct a deterministic transition system whose labels correspond to the belief predicates involved in the scLDTL formula ϕ. In order to incorporate the state predicates into this discrete system, we relax all state predicates by mapping them to belief predicates, e.g., state predicate A is relaxed to the belief predicate $(-\sum_{s\in A} b(s))$ (i.e. $Pr[s \in A] > 0$) (line 5). We also create a mapping Ψ_F from each belief predicate to an atomic proposition (lines 3-7). Then, for each f appearing in the relaxed scLDTL formula, we calculate the level set $f(b) = 0$ in $Dist(S)$ and map it to a set of probability vectors in the belief simplex. Many useful belief predicates, such as inequalities over moments, have polytopic level sets that are readily calculated. The level sets induce a partition of the simplex. A general algorithm for producing this partition will likely require the use of geometric tools and direct evaluations of the functions f for points in the simplex. We take the quotient of the partition to form a transition system and label each state with $\Psi_F(f)$ for each f that was satisfied in the corresponding region (lines 13-24). We denote the region of the simplex corresponding to the state q_j in the transition system as $Reg(q_j)$.

The condition in line 22 used to create transitions in the quotient involves a notion

of reachability that we make precise now.

Definition 4.6 (Reachability)**.** We say a state q_k is reachable from state q_m if beginning from any belief state in $Reg(q_m)$ there exists a sequence of actions and observations in $POMDP$ such that sequentially applying (3.2) will drive the system to a belief state associated with a belief state in $Reg(q_k)$. □

Determining the reachability relationship between states is a non-trivial process. In this work, we assume that all states are self-reachable and all state pairs corresponding to neighboring regions in the belief simplex are mutually reachable. We make this liberal assumption because if we observe a transition during monitoring that we did not assume to exist, FTS would be invalid. Allowing all possible transitions does not weaken our approach if a reachability relationship is false. If a transition cannot be made in FTS, we will never observe it during monitoring. This assumption will have to be relaxed in model checking or synthesis. Further, each transition is annotated with a virtual action rather than a collection of action/observation sequences.

Feasibility checking of a scLDTL formula proceeds according to Algorithm 4.2. From ϕ, we create an scLTL formula ϕ' by replacing every predicate in ϕ with its image in the mapping Ψ_F (lines 3 - 5). We then construct the automaton $\mathcal{A}_{\phi'}$ and form $\mathcal{P}_{\phi'}$, the synchronous product of FTS (from Algorithm 4.1) and $\mathcal{A}_{\phi'}$. The sequence $b^{0:t}$ is translated into the corresponding word $\alpha^{0:t}$ in the input language of $\mathcal{P}_{\phi'}$ (lines 9 - 14). We use $\mathcal{P}_{\phi'}$ to perform scLTL verification of ϕ'. If verification succeeds, Algorithm 4.2 returns a deterministic transition system DTS used in probabilistic acceptance checking to describe the time evolution of the satisfaction of belief predicates. DTS is a simple,"linear" transition system whose action set is a singleton and whose only possible run is $q_0 \dots q_t$ where $L_D(q_k) = L_F(q|b^k \in Reg(q))$.

If verification fails, then we do not proceed to probabilistic acceptance checking, as

Algorithm 4.1 Construct a transition system used to check a necessary condition for $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}] > 0$

function feasibilitySystemConstruct(ϕ, S,b^0)
predicateSet := $\emptyset$; $j := 1$; $\Pi = \emptyset$;
for all predicates $\in \phi$ **do**
  if predicate $\notin F_S$ **then**
    predicate := $(-\sum_{s \in \text{predicate}} b(s))$
  predicateSet := predicateSet $\cup$ predicate
  Ψ_F(predicate) := π_j
  $\Pi := \Pi \cup \pi_j$; $j := j + 1$;
for all $f \in$ predicates **do**
  calculate level set$f(b) = 0$
use the probability vector representation of the level sets to partition the belief simplex
$Q_F := \emptyset$; m := 1;
for all regions $\in$ partition **do**
  $Q_F := Q_F \cup \{q_m\}$;
  $L_F(q_m) := \{\Psi(f)|f(b) < 0 \ \forall b \in \text{region}\}$
  $Reg(q_m) :=$ region
  if $b^0 \in$ region **then**
    $q_0 := q_m$;
  $m := m + 1$
$Act_F := \emptyset$; $Trans_F := \emptyset$
for all $q_m, q_k \in Q_F^2$ **do**
  if q_k is reachable from q_m **then**
    $Act_F := Act_F \cup \{a_{mk}\}$
    $Trans_F := Trans_F \cup \{(q_m, a_{mk}, q_k)\}$
return $FTS = (Q_F, q_0, Act_F, Trans_F, \Pi_F, L_F), \Psi_F$

Algorithm 4.2 Returns a transition system that describes the time evolution of belief predicate satisfaction if the necessary condition for $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}] > 0$ holds

function feasibilityCheck($b^{0:t}, \phi, S$)
FTS, Ψ_F := feasibilitySystemConstruct(ϕ, S, b^0)
$\phi' := \phi$
for all predicates $\in \phi'$ **do**
  replace predicate in ϕ' with Ψ_F(predicate);
Construct the finite state automaton (FSA) $\mathcal{A}_{\phi'}$ that only accepts words satisfying ϕ'.
$\mathcal{P}_{\phi'} = FTS \times A_{\phi'}$
currentState := q_0; currentIndex := 0; $k := 1$
$Q_D := \emptyset$; $Act_D = \{a_0\}$; $Trans_D = \emptyset$; $\Pi_D := L_F(q_0)$
for $i = 1$ to t **do**
  if $b^i \notin Reg$(currentState) **then**
    currentState := q_j such that $b^i \in Reg(q_j)$
    $\Pi_D := \Pi_D \cup L_F$(currentState)
    currentIndex := j;
  $Q_D := Q_D \cup q_k$
  $Trans_D := Trans \cup (q_{k-1}, a_0, q_k)$
  $L_D(q_k) = L_F$(currentState)
  $\alpha^i := a_{\text{currentIndex,nextIndex}}$
  currentIndex := nextIndex
if $\alpha^{0:t-1}$ produces an accepting run on $\mathcal{P}_{\phi'}$ **then**
  return $DTS = (Q_D, q_{0,D}, Act_D, Trans_D, \Pi_D, L_D)$
return False

failure means that $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}] = 0$. Due to the mapping of state predicates to belief predicates, Algorithm 4.2 checks for the existence of at least one sample path $s^{0:t}$ such that $(s^0, b^0) \dots (s^t, b^t) \models \phi$ and $\prod_{i=0}^{t} b^i(s^i) > 0$. The positivity of the product is a necessary but not sufficient condition for $Pr[s^{0:t}|a^{0:t-1}, o^{1:t}, b^0] > 0$.

We illustrate Algorithms 4.1 and 4.2 in the following example.

Example 4.1. Consider the multiple hypothesis testing POMDP MHT given in Section 4.2 with scLDTL specification (4.3). Figure 4·2(a) shows the partitioning of the belief simplex from the belief predicates in (4.3) resulting from Algorithm 4.1. The predicates involving maximum likelihood (red) and specified entropy level (blue) partition the simplex into six regions corresponding to discrete states $q_i, i \in \{1, \dots, 6\}$. Each red curve is a level set $b([s_i, s_O]) = b([s_j, s_0]$ for $i \neq j$ and each blue curve is part of the level set $H(b) = 0.8$ bits. From this partition, we can execute Algorithm 4.1, lines 13-24 to form the transition system FTS shown in Figure 4·2(b). A state in FTS is labeled with proposition $\pi_j, j \in \{1, 2, 3\}$ if s_j is the most likely hypothesis according to any probability vector in the corresponding region. A state in FTS is labeled with proposition π_4 if the entropy of any probability vector in that region is less than 0.8 bits.

The green curve in Figure 4·2(a) represents a single, randomly generated execution of MHT. The observation likelihood parameters were $p_1 = 0.25, p_2 = 0.5, p_3 = 0.75$. Observations were generated with parameter p_1. Each point in the curve is the probability vector representation of the belief state b^i resulting from incorporating i observations via (3.2). The transition system DTS resulting from executing Algorithm 4.2 on the given sequence of belief states is shown in Figure 4·2(c). For the first three observations seen, the trajectory stays in $Reg(q_1)$. Thus the first three states in DTS are labeled with π_1. After the fourth measurement, the trajectory has gathered

enough information to enter $Reg(q_4)$. Thus the fourth (and final) state in DTS is labeled with both π_1 and π_4.

□

4.4.2 Probabilistic acceptance checking

If Algorithm 4.2 succeeds, we proceed to probabilistic acceptance checking. In this section, we use labeled Markov decision processes (LMDPs) and labeled Markov chains (LMCs) as abstractions to describe the probabilistic time evolution of the hidden states of the system. An LMDP is an MDP in which the states of the system are labeled with atomic propositions. An LMDP is given as a tuple $LMDP = (S, p_S^0, P, Act, AP, L)$ where S, P, and Act are as defined for a POMDP. The pmf over states p_S is not conditioned on observations. AP is a set of atomic propositions and $L : S \to 2^{AP}$ maps states to propositions. A labeled Markov Chain (LMC) is an LMDP without actions and is given as a tuple $LMC = (S, p_S^0, P, AP, L)$ where S, p_S^0, AP, and L are as defined for the LMDP and the probabilistic transition relationship P is not parameterized by actions.

We begin probabilistic acceptance checking by creating a mapping $\Psi_{sp} : 2^S \to \Pi_r$ that maps state predicates to atomic propositions in the set Π_r. This construction is similar to the construction of Ψ_F. The scLDTL formula ϕ is mapped to a scLTL formula ϕ'' by applying the mapping Ψ_F to the belief predicates and the mapping Ψ_{sp} to the state predicates appearing in ϕ. An FSA is created from ϕ''. Next, we enumerate all of the sample paths consistent with the given execution of $POMDP$. We do this by creating a labeled Markov chain LMC for each possible initial state s^0 such that $b^0(s^0) > 0$. LMC has a tree-like structure with root s^0. Each node s^i has as children any state s^{i+1} such that $P(s^i, a^i, s^{i+1}) > 0$ and $h(s^{i+1}, a^i, o^{i+1}) > 0$. Each state s in the tree is labeled with $\{\Psi_{sp}(A) | A \text{ appears in } \phi,\ s \in A\}$. The transition

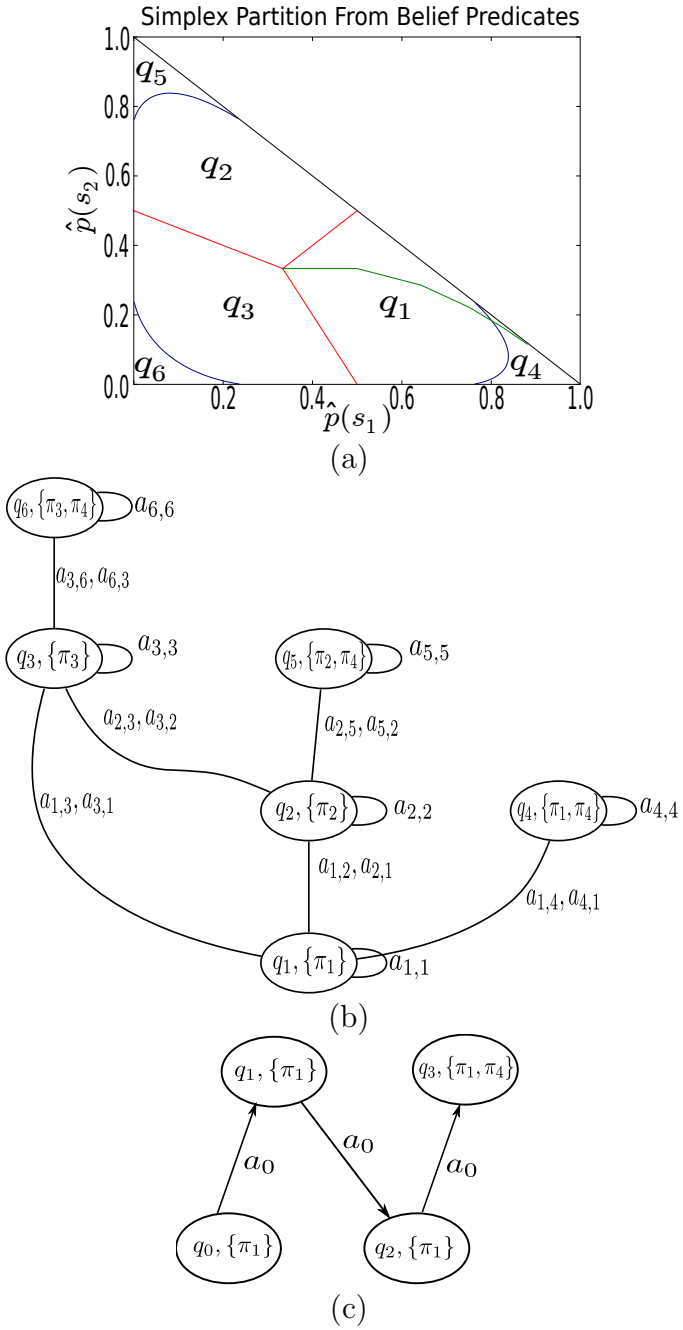


Figure 4·2: (a) The belief simplex for b_{S_h} partitioned according to the belief predicates used in (4.3). The red lines divide the simplex into three regions corresponding to the most likely hypothesis. The blue curves are the level sets $H(b_S) = 0.8$ bits. The green curve shows the probability trajectory corresponding to a sequence of belief states from a randomly generated execution of MHT. (b) The transition system FTS constructed by taking the quotient of the partition shown in (a). (c) The transition system DTS that results from applying Algorithm 4.2 to the given belief state sequence and FTS.

probability between states s^i, s^{i+1} is given by

$$P_{LMC}(s^i, s^{i+1}) = \frac{Pr[o^{i+1:t}|s^{i+1}, a^{i+1:t}]P(s^i, a^i, s^{i+1})}{\sum_{s\in S} Pr[o^{i+1:t}|s, a^{i+1:t}]P(s^i, a^i, s)}.$$

The details of this calculation can be found in (Briers et al., 2010). We construct $LMDP$, the synchronous product of LMC and DTS, which encapsulates the time evolution of both the satisfaction of state predicates (from LMC) and belief predicates (from DTS). A state in the ith level of $LMDP$ is labeled with atomic propositions associated with the belief predicates satisfied by b^i and state predicates satisfied by a state s^i that is reachable from state s^0 given the first i actions and observations.

Since the action set Act_{DTS} is a singleton, there is no notion of choice in the evolution of DTS and thus no choice in the evolution of $LMDP$. We create another labeled Markov chain LMC_P from $LMDP$ by removing the action set and using the probabilistic transition relationship $P_{LMC_P}(s, s') = P_{LMDP}(s, a_0, s')$. We then form $\mathcal{M}$, the synchronous product of LMC_P and $\mathcal{A}_{\phi''}$. We perform model checking on LMC_P to find the set of all accepting runs $Acc(\phi'')$ on $\mathcal{M}$ of length $t+1$. Each run in $Acc(\phi'')$ corresponds to a sample path that satisfies ϕ when paired with $b^{0:t}$. For each run $r^{0:t}$ in $Acc(\phi'')$, let $s^{0:t}$ be the corresponding sample path over LMC_P. We calculate $Pr[s^{0:t}|a^{0:t-1}, o^{1:t}] = Pr[s^0|a^{0:t-1}, o^{0:t}] \prod_{i=1}^{t} P_{LMC_P}(s^i, s^{i+1})$ (Briers et al., 2010) and add it to the acceptance probability $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}]$. By enumerating over all possible sample paths, we calculate the exact value of $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}]$.

4.5 Case Study

A proposed use of mobile robots is to perform rescue operations in areas that are too hazardous for human rescuers. A robot is deployed to a location such as an

office building or school after a natural disaster and is tasked with finding all human survivors in the environment and with moving any immobilized survivors to safe areas. The robot must learn survivor locations and safety profile of the building on-line by processing noisy measurements from its sensors. The combination of on-line estimation and time-sensitive decision-making indicates that scLDTL is a good framework for describing the mission specification at a high level.

4.5.1 Model

For simplicity, we consider a rescue robot acting in a two room environment. We model the robot as a POMDP $Rescue = (S, b^0, P, Act, Obs, h)$. The state of the system is given by a vector $[s_q, s_O, s_{1,e}, s_{2,e}, s_{1,s}, s_{2,s}]$ in the state space $S = \{1, 2\} \times \{0, 1\}^5$. The element s_q corresponds to the room in which the robot currently resides and $s_O \in \{0, 1\}$ corresponds to whether ($s_O = 1$) or not the robot is carrying an object($S_O = 0$). The elements $s_{i,e} \in \{0, 1\}$ correspond to safety, i.e. if $s_{i,e} = 1$, then room i is safe to be occupied by a human. The elements $s_{i,s} \in \{0, 1\}$ correspond to survivor presence, i.e. if $s_{i,s} = 1$, a survivor is in room i.

The robot can stay in its current room and measure its surroundings, switch to the other room, pick up an object, or put down an object. Here we assume the motion model of the robot is deterministic, the safety of the environment is static, and the survivor locations change only if the robot moves a survivor. If the robot attempts to move a survivor, it fails with some probability p_{fail}.

If the robot takes action $Stay$, its sensors return observations in the set $Obs = \{0, 1\}^2$. The elements of Obs are binary reports of the safety and survivor occupancy of the current room. The sensor is parameterized by two independent false alarm and correct detection rates.

4.5.2 Problem statement

For convenience we establish the shorthand $b_j(\sigma) = \sum_{\{s \in S | s_j = \sigma\}} b(s)$ where s_j is a component of an element of S. We wish to find and move all of the survivors in the given area to safe regions. In order for the robot to be reasonably sure that this condition is met, it must be fairly certain about the state of the environment. Therefore, we want the entropy of our estimate to be low, i.e.

$$\forall i \in \{1, 2\}\ H(b_{i,e}) < h_1,\ H(b_{i,s}) < h_2. \tag{4.5}$$

Survivor safety is time-critical. We thus require "If the robot is confident a survivor is in an unsafe location, move it to a safe location". We encode confidence by saying "with a certain probability".

The statement that describes the rescue robotics application is "Explore the environment and if the robot is in a state where it is sure with probability p_1 there is a survivor and with probability p_2 the state is unsafe, pick up the survivor, move to the other room and deposit the survivor. Perform these actions until (4.5) and any identified survivors are in safe regions". The above statement is encoded in the scLDTL formula $\phi_1 \mathcal{U} \phi_2$ where

$$\begin{aligned} \phi_1 = \quad & (\{s|s_q = j\} \wedge (p_1 - b_j(s)) \wedge (p_2 - b_{j,e}(0))) \\ & \Rightarrow (\bigcirc(\{s|s_O = 1\} \mathcal{U} \{s|s_q \neq j\}) \wedge \bigcirc\{s|s_O = 0\} \\ \phi_2 = \quad & \textstyle\bigwedge_{i \in \{1,2\}} (H(b_{i,e}) - h_1) \wedge (H(b_{i,s}) - h_2) \\ & \wedge(\{s|s_{i,e} = 1\}) \wedge \{s|s_{i,s} = 1\}) \vee \{s|s_{i,s} = 0\}) \end{aligned} \tag{4.6}$$

The formula ϕ_1 encodes "if the robot is in a state where it is sure with probability p_1 there is a survivor ($\{s|s_q = j\} \wedge (p_1 - b_j(s))$) and with probability p_2 the state is unsafe ($p_2 - b_{j,e}(0)$), pick up the survivor ($\{s|s_O = 1\}$), move to the other room

$\{s|s_q \neq j\}$) and deposit the survivor ($\{s|s_O = 0\}$)." The formula ϕ_2 encodes "Perform these actions until (4.5) ($\bigwedge_{i\in\{1,2\}}(H(b_{i,e}) - h_1) \wedge (H(b_{i,s}) - h_2)$) and any survivors are in safe regions (($\{s|s_{i,e} = 1\}$) $\wedge$ $\{s|s_{i,s} = 1\}$) $\vee$ $\{s|s_{i,s} = 0\}$))."

Due to the time sensitive nature of survival, we consider the following time-constrained optimization problem.

$$\max_{a^{0:t}} E_{\{o^{1:t}\}}[Pr[\phi_1 \mathcal{U} \phi_2 | b^{0:t}, a^{0:t-1}, o^{1:t}]] \tag{4.7}$$

4.5.3 Acceptance checking

We consider two separate strategies: time share and entropy cutoff. In the time share strategy with parameter a, the robot switches rooms every $\lceil \frac{t}{a} \rceil$ observations. In the entropy cutoff strategy with parameters h_3, h_4, ρ, the robot switches rooms when the entropy of the estimate of the safety and survivor presence of the current room dips below h_3 and h_4, respectively. If the estimates of both rooms are of the specified certainty, the agent must wait ρ time units before switching. Both strategies include the reactive behavior of attempting to pick up survivors when they are found.

The results from 250 Monte Carlo trials of length $t = 16$ are shown in Figure 4·3. The control strategy parameters were parameter $a = 3, h_3 = h_4 = 0.3$, and $\rho = 2$. Further simulation parameters are given in the caption of Figure 4·3. Here we use $Pr[\phi]$ as shorthand for the statistic formed from samples of $Pr[\phi|b^{0:t}, a^{0:t-1}, o^{1:t}]$ collected from the trials. For both methods, there are clusters of points around the lines $Pr[\phi] = 1$ and $Pr[\phi] = 0$. This is because by making the entropy of the belief state a temporal goal in the scLDTL formula, the probability calculation sets the acceptance probability to 0 for executions after which the characterization of the environment is ambiguous, i.e. when the probability is close to the center of the interval $[0, 1]$.

Method	$E[Pr[\phi]]$	$var(Pr[\phi])$	$E[H(b^t)]$	$var(H(b^t)$	success rate	$R(Pr[\phi], H(b^t))$
Timeshare	0.855	0.115	0.366 bits	0.150 bits^2	0.86	-0.547
Entropy Threshold	0.992	0.004	0.338 bits	0.034 bits^2	0.916	-0.341

Table 4.1: Statistics from 250 Monte Carlo trials of the two-room rescue robotics simulation.

The statistics resulting from our simulations are shown in Table 4.1. The statistic $R(Pr[\phi], H(b^t))$ is the Pearson's R correlation coefficient between the two variables. The success rate is given as the number of trials such that at time $t = 16$, all survivors were safe divided by the total number of trials. Note that the entropy cutoff method performs better in terms of acceptance probability, expected terminal entropy, and success rate. This matches intuition, as this method will drive the robot to stay in a room longer if the particular observation sequence it observes does not lead to any strong conclusions or it will move to the other room if it has already obtained a good estimate. This is in contrast to the time share method, which ignores estimate quality in its decision policy.

Further, note that for both methods, the correlation coefficient is weakly negative. This weakness is due to the clustering of points at varying entropies around $Pr[\phi] = 0$ and $Pr[\phi] = 1$. This negative correlation and the relative closeness of the average acceptance probability of the two methods to their respective success rates suggests that for some appropriately-defined scLDTL formulae, the probability $Pr[\phi | b^{0:t}, a^{0:t-1}, o^{1:t}]$ is an appropriate metric for the dual consideration of estimate quality (uncertainty) and system performance.

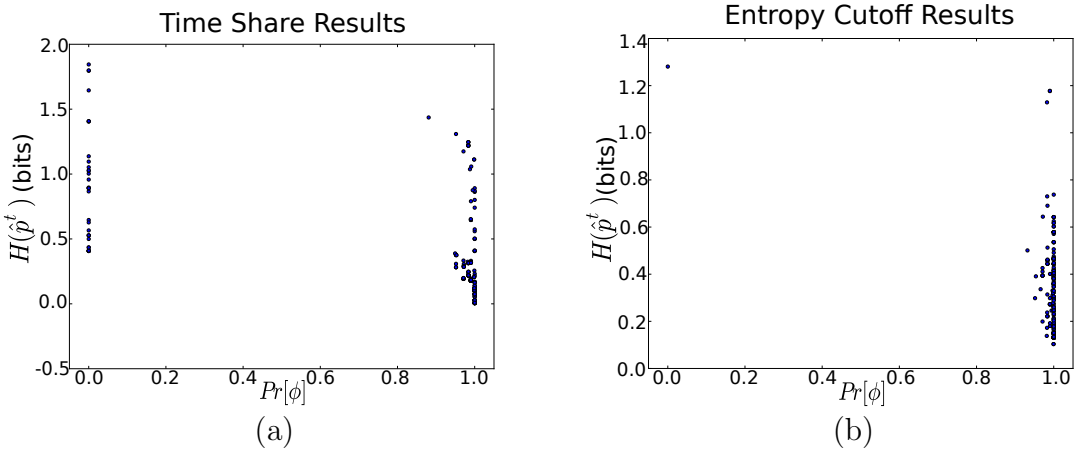


Figure 4·3: Scatter plots showing the results of 250 Monte Carlo trials of the two room rescue robot POMDP under policy (a) time share and (b) entropy cutoff. The parameters used in (4.6) are $h_1 = h_2 = 0.375$ and $p_1 = 0.9$, $p_2 = 0.25$. The probability that an agent fails to pick up a survivor was $p_{fail} = 0.4$. The false alarm rates for safety and survivor were both 0.1. The correct detection rates for safety and survivor were 0.8 and 0.9, respectively.

Chapter 5

Signal Temporal Logic

In the previous chapters, we have focused on verifying and enforcing propositional temporal logic formulae on systems that can be modeled as discrete, graph-like representations. In the following chapters, we investigate two problems (TL inference and reinforcement learning for satisfying TL specifications) that are defined with respect to continuous systems. For this reason, we use a recently developed predicate temporal logic, called Signal Temporal Logic (STL) (Maler and Nickovic, 2004). In contrast to LTL, which can specify properties involving Boolean propositions and abstract time, STL can specify properties over continuous systems that include bounds on physical states and bounded time intervals. STL comes equipped with a recursively-defined robustness degree, that is, a continuous measure of how well a given execution of a continuous system satisfies or violates a given STL execution. Recent papers have used this robustness degree to guide parameter estimation problems for cyber-physical systems (Jin et al., 2013) and to control systems with continuous state space via model predictive control (Raman et al., 2014; Raman et al., 2015).

Signal temporal logic is also closely related to metric temporal logic (MTL) (Koymans, 1990), an extension of LTL that has interval time semantics, but lacks bounds on physical states. MTL has been used to verify and monitor models of complex cyber-physical systems (Dokhanchi et al., 2014; Abbas et al., 2014).

In this chapter, we give some preliminary definitions that are necessary for Chap-

ters 6 and 7.

Definition 5.1 (Signal)**.** Given two sets A and B, $\mathcal{F}(A, B)$ denotes the set of all functions from A to B. Given a time domain $\mathbb{R}^+ := [0, \infty)$ (or a finite prefix of it), a *continuous-time, continuous-valued signal* is a function $s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n)$. We use $s(t)$ to denote the value of signal s at time t, and $s[t]$ to denote the suffix of signal s from time t, i.e. $s[t] = \{s(\tau) | \tau \geq t\}$. We use x_s to denote the one-dimensional signal corresponding to the variable x of the signal s. □

Definition 5.2 (Signal Temporal Logic)**.** *Signal temporal logic* (STL) (Maler and Nickovic, 2004) is a temporal logic defined over signals.

The *syntax* of STL is inductively defined as

$$\phi := \mu | \neg\phi | \phi_1 \vee \phi_2 | \phi_1 \wedge \phi_2 | \phi_1 \mathcal{U}_{[a,b)} \phi_2, \tag{5.1}$$

where $[a, b)$ is a time interval, μ is a *numerical predicate* in the form of an inequality $g_\mu(s(t)) \sim c_\mu$ such that $g_\mu \in \mathcal{F}(\mathbb{R}^n, \mathbb{R})$, $\sim \in \{<, \geq\}$, and c_μ is a constant.

The *semantics* of STL is defined recursively as

$$\begin{array}{rll} s[t] \models \mu & \text{iff} & g_\mu(s(t)) \sim c_\mu \\ s[t] \models \neg\phi & \text{iff} & s[t] \not\models \phi \\ s[t] \models \phi_1 \wedge \phi_2 & \text{iff} & s[t] \models \phi_1 \text{ and } s[t] \models \phi_2 \\ s[t] \models \phi_1 \vee \phi_2 & \text{iff} & s[t] \models \phi_1 \text{ or } s[t] \models \phi_2 \\ s[t] \models \phi_1 \mathcal{U}_{[a,b)} \phi_2 & \text{iff} & \exists t' \in [t + a, t + b) \\ & & \text{ s. t. } s[t'] \models \phi_2, s[t''] \models \phi_1 \\ & & \forall t'' \in [t + a, t'). \end{array} \tag{5.2}$$

□

We also use the constructed temporal operators $\Diamond_{[a,b)} \phi = \top \mathcal{U}_{[a,b)} \phi$ (read "eventually ϕ"), where $\top$ is the symbol for Boolean constant True, and $\Box_{[a,b)} \phi = \neg \Diamond_{[a,b)} \neg\phi$ (read "always ϕ"). In plain English, the semantics of $\Diamond_{a,b} \phi$ means "within a and b

time units in the future, ϕ is true," $\Box_{[a,b)}\phi$ means "for all times a and b time units in the future ϕ is true," and $\phi_1 U_{[a,b)}\phi_2$ means "There exists a time c between a and b time units in the future such that ϕ_1 is true until c and ϕ_2 is true at c."

A signal s satisfies an STL formula ϕ if $s[0] \models \phi$. The *language* of an STL formula ϕ, $\mathcal{L}(\phi)$, is the set of all signals that satisfy ϕ, namely $\mathcal{L}(\phi) = \{s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n) | s \models \phi\}$. Given formulae ϕ_1 and ϕ_2, we say that ϕ_1 and ϕ_2 are *semantically equivalent* , i.e., $\phi_1 \equiv \phi_2$, if $\mathcal{L}(\phi_1) = \mathcal{L}(\phi_2)$.

Example 5.1 (STL Formula)**.** Consider the signal s shown in Figure 5·1. s satisfies the formula

$$\phi_g = \Box_{[0,3)}(\Diamond_{[0,2)}(s > 0.8) \wedge \Diamond_{[0,2)}(s < -0.8))$$

In plain English, ϕ_1 means "For every point in time t from 0s to 3s, within 2s in the future (i.e. during the interval $[t, t+2)$), the value of s exceeds 0.8 and dips below -0.8." This effectively captures the oscillatory behavior of s. In contrast, s does not satisfy

$$\phi_b = \Diamond_{[0,3)}(\Box_{[0,2)}(s > 0.8) \vee \Box_{[0,2)}(s < -0.8)),$$

which in plain English means "Within 3s, s remains above 0.8 for the next 2 s or s remains below -0.8 for the next 2 s. □

Remark 5.1. Because STL is a predicate temporal logic and has interval-based time semantics, it is in general not appropriate to construct an automaton from an STL formula. When determining whether or not a given signal satisfies an STL formula, the recursive semantics are used directly to calculate its satisfaction. □

Definition 5.3 (Parametric STL)**.** *Parametric signal temporal logic* (PSTL) (Asarin et al., 2012) is an extension of STL where c_μ or the endpoints of the time intervals $[a, b)$ are parameters instead of constants. We denote them as *scale* parameters $\pi =$

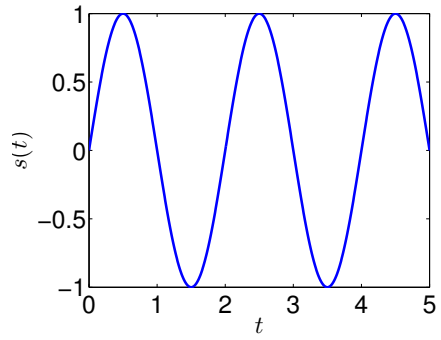


Figure 5·1: Simple sinusoid used in Example 5.1.

$[\pi_1, ..., \pi_{n_\pi}]$, and *time* parameters $\tau = [\tau_1, ..., \tau_{n_\tau}]$, respectively. They range over their respective hyper-rectangular domains $\Pi \subset \mathbb{R}^{n_\pi}$ and $T \subset \mathbb{R}^{n_\tau}$. A full parameterization is denoted by $\theta = [\pi, \tau]$ with $\theta \in \Theta = \Pi \times T$. The syntax and semantics of PSTL are the same as those for STL. □

To avoid confusion, we will use ϕ to refer to an STL formula and φ to refer to a PSTL formula. A *valuation* v is a mapping that assigns real values to the parameters appearing in a PSTL formula. Each valuation v of a PSTL formula φ induces an STL formula ϕ_v where each parameter in φ is replaced with its image in v. For example, given $\varphi = (x_s \geq \pi_1)\mathcal{U}_{[0,\tau_1)}(y_s \geq \pi_2)$ and $v([\pi_1, \pi_2, \tau_1]) = [0, 4, 5]$, we have $\phi_v = (x_s \geq 0)\mathcal{U}_{[0,5]}(y_s \geq 4)$.

Definition 5.4 (Robustness Degree)**.** The *robustness degree* of a signal s with respect to an STL formula ϕ at time t is given as $r(s, \phi, t)$, where r can be calculated recursively via the *quantitative semantics* (Fainekos and Pappas, 2009; Donzé and

Maler, 2010)

$$\begin{aligned} r(s, \mu_{\geq}, t) &= g_\mu(s(t)) - c_\mu \\ r(s, \mu_{<}, t) &= c_\mu - g_\mu(s(t)) \\ r(s, \neg\phi, t) &= -r(s, \phi, t) \\ r(s, \phi_1 \wedge \phi_2, t) &= \min(r(s, \phi_1, t), r(s, \phi_2, t)) \\ r(s, \phi_1 \vee \phi_2, t) &= \max(r(s, \phi_1, t), r(s, \phi_2, t)) \\ r(s, \phi_1 \mathcal{U}_{[a,b)} \phi_2, t) &= \sup_{t' \in [t+a, t+b)} (\min(r(s, \phi_2, t'), \\ & \quad \inf_{t'' \in [t, t')} r(s, \phi_1, t''))) \end{aligned}$$

where $\mu_{\geq}$ is a predicate of the form $g_\mu(s(t)) \geq c_\mu$ and $\mu_{<}$ is a predicate of the form $g_\mu(s(t)) < c_\mu$.

We use $r(s, \phi)$ to denote $r(s, \phi, 0)$. A *signed distance* from a point $x \in X := \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n)$ to a set $S \subseteq X$ is defined as

$$\text{Dist}_\rho(x, S) := \begin{cases} -\inf\{\rho(x, y) | y \in cl(S)\} & \text{if } x \notin S \\ \inf\{\rho(x, y) | y \in X \setminus S\} & \text{if } x \in S \end{cases} \tag{5.3}$$

with $cl(S)$ denoting the closure of S, ρ is a metric defined as

$$\rho(s, s') = \sup_{t \in T}\{d(s(t), s'(t))\}, \tag{5.4}$$

and d corresponds to the metric defined on the domain $\mathbb{R}^n$ of signal s. It has been shown in (Fainekos and Pappas, 2009) that $r(s, \phi)$ is an under-approximation of $\text{Dist}_\rho(s, \mathcal{L}(\phi))$. □

The robustness degree thus gives a measure of how robustly a signal either satisfies or violates the given STL formula. If $r(s, \phi)$ is large and positive (negative), then s satisfies (violates) ϕ and a large perturbation to s would be required in order for the resulting signal s' to violate (satisfy) ϕ. If $r(s, \phi) \approx 0$, then if even a small perturbation is applied to s, whether or not s' satisfies ϕ is unpredictable.

Example 5.1 (Cont'd)**.** If we calculate the robustness degree of s with respect to ϕ_g

and ϕ_b, we see that $r(s, \phi_g) = 0.2$, while $r(s, \phi_b) = -1.8$. This indicates that a small perturbation to s, e.g. a change in amplitude or frequency, could cause it to violate ϕ_g, while s would have to change by a large amount in order to stay above 0.8 or below -0.8 for 2 seconds. □

Definition 5.5 (Horizon Length)**.** Similar to (Dokhanchi et al., 2014), let $hrz(\phi)$ denote the *horizon length* of an STL formula ϕ. The horizon length is the required number of samples to resolve any (future or past) requirements of ϕ. The horizon length can be computed recursively as follows (adopted from (Dokhanchi et al., 2014)):

$$\begin{aligned} hrz(\mu) &= 0, \\ hrz(\neg\phi) &= hrz(\phi), \\ hrz(\phi_1 \vee \phi_2) &= \max\{hrz(\phi_1), hrz(\phi_2)\}, \\ hrz(\phi_1 \wedge \phi_2) &= \max\{hrz(\phi_1), hrz(\phi_2)\}, \\ hrz(\phi_1 U_{[a,b)} \phi_2) &= \max\{hrz(\phi_1) + b - 1, hrz(\phi_2) + b\} \\ hrz(\ \diamondsuit_{[a,b))}\phi) &= hrz(\phi) + b \\ hrz(\ \square_{[a,b))}\phi) &= hrz(\phi) + b \end{aligned} \tag{5.5}$$

□

Example 5.1 (Cont'd)**.** The horizon length for ϕ_g can be calculated as

$$hrz(\phi_g) = 3 + \max(2 + 0, 2 + 0) = 5 \tag{5.6}$$

□

Chapter 6

Learning Specifications from Data

In this chapter, we consider the problem of learning signal temporal logic formulae from data (Kong et al., 2014; Jones et al., 2014; Kong et al., 2015). That is, we address problems in which we do not have access to an explicit model of a system but only the outputs from the system. These methods are therefor applicable in situations where we want to learn high-level behaviors described as temporal logic fomulae from complex systems such as autonomous or human-driven vehicles, where obtaining an accurate model is difficult. Our techniques are also amenable to cases in which models are too large or complex to be analyzed with traditional tools. In this chapter, we use this problem formulation to address problems in cyber-physical system security.

In Section 6.1, we use off-line supervised learning to learn formulae to distinguish desirable behaviors from undesirable behaviors. That is, we learn formulae from datasets where experts have labeled the data according to whether or not each trajectory represents a desirable or undesirable behavior. In Section 6.2, we use on-line supervised learning to learn a formula to classify desirable and undesirable behaviors. That is, instead of learning from a database, the on-line algorithm updates the formula as new data becomes available. Finally, in Section 6.3, we use unsupervised learning to learn formulae that distinguish anomalous behaviors from mainstream, "normal" behaviors. That is, we learn formulae from datasets where the data is un-

labeled. Each section is organized into subsections that define the problem under consideration, present the algorithm for its solution, and demonstrate the algorithm on case studies.

6.1 Supervised Learning of STL specifications

In this section, we address the problem of inferring a temporal logic formula that can be used to distinguish between desirable system behaviors, e.g. an airplane lands in some goal configuration on the tarmac, and undesirable behaviors, e.g. the airplane's descent is deemed unsafe. Moreover, in our approach, the inferred formulae can be used as predictive templates for either set of behaviors. This in turn can be used for on-line system monitoring, e.g. aborting a landing if the descent pattern is consistent with unsafe behavior. Since our procedure is automatic and unsupervised beyond the initial labeling of the signals, it is possible that it can discover properties of the system that were previously unknown to designers, e.g. changing the direction of banking too quickly will drive the airplane to an unsafe configuration.

This section also introduces inference parametric signal temporal logic (iPSTL), a fragment of PSTL, which we originally defined in (Kong et al., 2014; Kong et al., 2015). iPSTL is expressive enough to capture properties that are crucial to a wide range of applications. In Section 6.1.2, we show that we are able to build a directed acyclic graph (DAG) for all iPSTL formulae. The formulae in the DAG are organized according to how general they are such that if φ_2 is a child of φ_1, then the property described by φ_1 implies the property described by φ_2. This result enables us to formulate the supervised learning problem as an optimization problem over iPSTL formula structures and continuous parameterizations.

6.1.1 Problem Definition

We wish to construct a classifier that can separate outputs from a system behaving normally from outputs from a system behaving abnormally. Here, we consider the case in which our inference procedure can learn from historical data that has been labeled according to whether or not it represents a normal behavior. More formally, we wish to solve Problem 6.1.

Problem 6.1. Let $\{x_i\}_{i=1}^{M}$ be a set of trajectories generated by $\mathcal{S}$. Let s_i be the observed output signal associated with x_i and p_i be the corresponding label assigned by expert or database knowledge. $p_i = 1$ if s_i represents a normal behavior and $p_i = -1$ if s_i represents an anomalous behavior. From the pairs $\{(s_i, p_i)\}_{i=1}^{M}$, find an iSTL formula (defined in Section 6.1.2) ϕ_N such that the misclassification rate

$$MR_{\mathcal{L}}(\{(s_i, p_i)\}_{i=1}^{M}, \phi_N) = \frac{FA_L + MD_L}{M} \tag{6.1}$$

is minimized, where

$$FA_L = |\{s_i | s_i \not\models \phi, p_i = 1)\}|$$

is the number of false alarms (signals improperly classified as anomalous) and

$$MD_L = |\{s_i | s_i \models \phi, p_i = -1\}|$$

is the number of missed detections (signals improperly classified as normal). This problem can be adapted to the problem of constructing a formula ϕ_A that describes only those outputs from abnormal systems. □

Here, we give a motivating scenario that illustrates Problem 6.1.

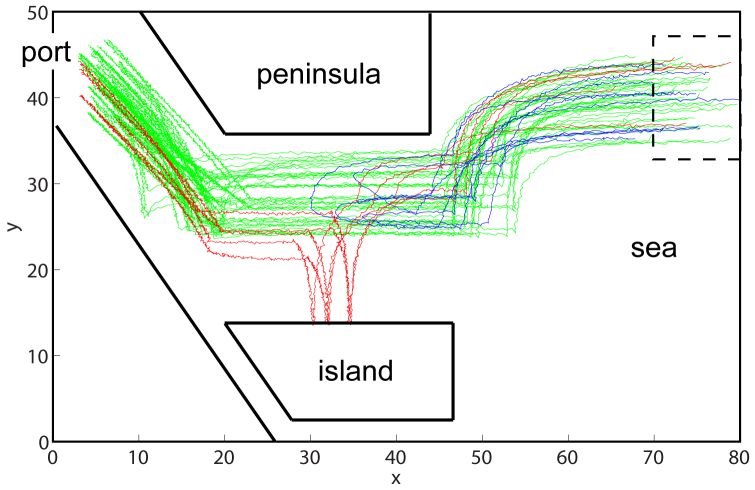


Figure 6·1: A naval surveillance example. Trajectories of vessels behaving normally are shown in green. The red trajectories represent possible human trafficking scenarios and the blue trajectories represent possible terrorism scenarios. The layout resembles Boston harbor.

Example 6.1 (Naval Surveillance)**.** In maritime surveillance (Kowalska and Peel, 2012), the Automatic Identification System (AIS) enables law enforcement authorities to collect data at regular intervals on a large number of ships. The available data includes the vessels' locations, courses, speeds, and destinations. This information can be used to uncover security threats and suspicious activities such as drug smuggling, human trafficking, arms trading, or terrorism. However, high volumes of traffic make manual inspection of the collected data for anomalous behavior time-consuming and error-prone. Thus there is a need for systems that automate the process of detecting and responding to anomalous events. Consider the academic example shown in Figure 6·1.

Normal Behavior A vessel behaving normally (shown in green in Figure 6·1) approaches from the sea until it reaches the narrow passage between the peninsula and the island. Then, it heads directly towards the port.

Anomalous Behavior Consider two scenarios modified from (Kowalska and Peel, 2012) that may indicate illicit actions. In the first scenario, a vessel (shown in red in

Figure 6·1) deviates to the island. This may indicate a human trafficking scenario in which the vessel initially follows a normal track, then heads to the shore to pick up people before returning to its original path. In the second scenario, a vessel (shown in blue in Figure 6·1) approaches a ferry, loiters, and then quickly returns to the open sea. This behavior may indicate terroristic activity in which the vessel plants an explosive device on or near the ferry. □

This example requires us to learn a classifier that differentiates desired behaviors from undesirable behaviors. A single output of this system can have a large number of data points, which means that finding a classifier using traditional machine learning methods would require the definition of features. For example, we could use time to reach a point in the state space in the maritime example or frequency of oscillation in the brake example. However, these features or set of features must be defined by some expert with knowledge of the problem domain (or by visual data inspection). Our methods are able to solve each of these problems directly without using such user-defined features, thus minimizing the need for an expert.

6.1.2 Inference Parametric Signal Temporal Logic

In this chapter, we focus on a fragment of PSTL that we call *inference PSTL* (iPSTL). This fragment has a partial order over formulae that is essential for the formula structure search we perform in Section 6.2. The *syntax* of iPSTL is given as

$$\varphi ::= \ \diamondsuit_{[\tau_1,\tau_2)} \varphi_i \tag{6.2a}$$

$$\varphi_i ::= \ \diamondsuit_{[\tau_1,\tau_2)} \ell | \ \square_{[\tau_1,\tau_2)} \ell | \varphi_i \wedge \varphi_i | \varphi_i \vee \varphi_i \tag{6.2b}$$

where ℓ is a linear predicate of the form $(x_s \sim \pi)$ and x_s is a coordinate of the signal s. Since iPSTL is a fragment of PSTL, any valuation θ of an iPSTL formula induces an STL formula. We call the fragment of all such STL formulae inference STL (iSTL). The semantics of iSTL are the same as defined in Definition (5.2). Given an iPSTL formula φ and a valuation θ, we denote the corresponding iSTL formula as ϕ_θ.

Expressivity

iPSTL can be used to express a wide range of important system properties, such as

- Bounded-time invariance, e.g. $\diamondsuit_{[0,\tau_1)}(\square_{[\tau_2,\tau_3)}(y_s < \pi))$ ("There exists a time $t \in [0, \tau_1)$ such that y_s will always be less than π in $[t + \tau_2, t + \tau_3)$.")
- Reachability to multiple regions in the state space, e.g. $\diamondsuit_{[0,\tau_1)}(\diamondsuit_{[\tau_2,\tau_3)}(y_s \geq \pi_1) \vee \diamondsuit_{[\tau_2,\tau_3)}(y_s < \pi_2))$ ("There exists a time $t \in [0, \tau_1)$ such that eventually y_s is either less than π_1 or greater than π_2 from $t + \tau_2$ seconds to $t + \tau_3$ seconds.")

Example 6.1 (Naval Surveillance (contd.))**.** The normal vessel behavior can be described by the iSTL formula

$$\begin{aligned}\varphi_N \quad &= \diamondsuit_{[0,590)}(\square_{[0,200)}(y_s \geq 20) \wedge \square_{[0,200)}(y_s < 35) \\ &\quad \wedge \diamondsuit_{[0,350)}(x_s < 25))\end{aligned} \tag{6.3}$$

As shown in Figure 6·1, the two scale parameters related to y_s, 20 and 35, define the bounds of the normal traces corresponding to the narrow passage between the peninsula and the island. The scale parameter related to x_s, 25, defines the right boundary of the port. In plain English, this formula reads "There is a time t within [0, 590) such that the vessel's y coordinate should always be between 20 and 35 for the next 350 units and within 350 units the vessel will eventually reach an x coordinate that is less than 25". The human trafficking scenario (shown in red in Figure 6·1)

violates the conjunction of the first and second clauses while the terrorism scenario (shown in blue in Figure 6·1) violates the third clause. □

There are some temporal properties that cannot be described directly in iPSTL, namely,

- Concurrent eventuality, e.g. $\varphi_i = \ \diamondsuit_{[0,\tau_1)}((y_s < \pi_1) \wedge (x_s \geq \pi_2))$. ("Within τ_1 seconds, y_s is less than π_1 and x_s is greater than π_2 at the same time.")

- Nested "always eventually", e.g. $\varphi_i = \ \square_{[0,\tau_1)} \diamondsuit_{[\tau_2,\tau_3)} (y_s < \pi_1)$. ("At any time t in the next τ_1 seconds, y_s will be less than π_1 at some point in the interval $[t + \tau_2, t + \tau_3)$.")

The lack of concurrent eventuality means that we cannot directly specify that a trajectory will eventually reach some intersection of half-spaces in the state-space, though we can approximate such properties by specifying $\varphi_i = \ \diamondsuit_{[0,\tau_1)}(y_s < \pi_1) \wedge \diamondsuit_{[0,\tau_1)}(x_s > \pi_2)$.

The lack of nested "always eventually" limits the periodic properties that may be expressed, but we can approximate such properties by specifying $\varphi_i = \ \diamondsuit_{[\tau_2,\tau_3)}(y_s < \pi_1) \wedge \ldots \wedge \ \diamondsuit_{[\tau_2+n\epsilon,\tau_3+n\epsilon)}(y_s < \pi_1)$, that is by selecting n points in the interval $[0, \tau_1)$ ϵ apart and specifying that the property $\diamondsuit_{[\tau_2,\tau_3)}(y_s < \pi_1)$ is true at all points.

Properties of iPSTL

In this subsection, we fist define a partial order over $iPSTL$, the set of all iPSTL formulae. The formulae in $iPSTL$ can be organized in a directed acyclic graph (DAG) where a path exists from formula φ_1 to formula φ_2 iff φ_1 has a lower order than φ_2. Finally, for any parameterization, the robustness degree of a signal with respect to a formula $\phi_{1,\theta}$ is greater than with respect to $\phi_{2,\theta}$ if φ_1 has a higher order than φ_2. This

enables us to find an iSTL formula against which a signal is more robust by searching for a parameterization of an iPSTL formula that is further down the DAG.

Partial Orders Over iSTL and iPSTL

We define two relations $\preceq_S$ and $\preceq_P$ for iSTL formulae and iPSTL formulae, respectively.

Definition 6.1.

1. For two iSTL formulae ϕ_1 and ϕ_2, $\phi_1 \preceq_S \phi_2$ iff $\forall s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n), s \models \phi_1 \Rightarrow s \models \phi_2$, i.e. $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_2)$.

2. For two iPSTL formulae φ_1 and φ_2, $\varphi_1 \preceq_P \varphi_2$ iff $\forall \theta$, $\phi_{1,\theta} \preceq_S \phi_{2,\theta}$, where the domain of θ is $\Theta(\varphi_1) \cup \Theta(\varphi_2)$, the union of parameters appearing in φ_1 and φ_2.

□

Based on these definitions and the semantics of $iSTL$ and $iPSTL$, we have

Proposition 6.1. *Both $\preceq_S$ and $\preceq_P$ are partial orders.*

Proof. A *partial order* $\preceq$ is a binary relation that is reflexive, transitive and antisymmetric.

($\preceq_S$) *Reflexivity* $\phi_1 \preceq_S \phi_1$ is equivalent to $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_1)$, which is trivially true. *Transitivity* $\phi_1 \preceq_S \phi_2$ and $\phi_2 \preceq_S \phi_3$ is equivalent to $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_2)$ and $\mathcal{L}(\phi_2) \subseteq \mathcal{L}(\phi_3)$. It implies $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_3)$, which means $\phi_1 \preceq_S \phi_3$. *Antisymmetry* $\phi_1 \preceq_S \phi_2$ and $\phi_2 \preceq_S \phi_1$ is equivalent to $\mathcal{L}(\phi_1) \subseteq \mathcal{L}(\phi_2)$ and $\mathcal{L}(\phi_2) \subseteq \mathcal{L}(\phi_1)$. It implies $\mathcal{L}(\phi_1) = \mathcal{L}(\phi_2)$, which means $\phi_1 \equiv \phi_2$.

($\preceq_P$) Regardless of the relationship among formulae φ_1, φ_2 and φ_3, the relationship among their parameter sets $\Theta(\varphi_1)$, $\Theta(\varphi_2)$ and $\Theta(\varphi_3)$ can be generally represented as in

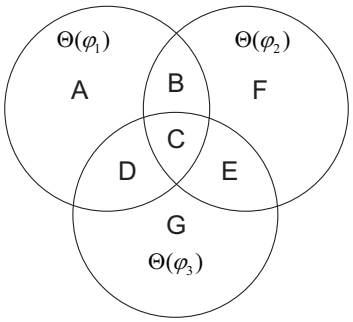


Figure 6·2: Relationship among $\Theta(\varphi_1)$, $\Theta(\varphi_2)$ and $\Theta(\varphi_3)$.

Figure 6·2. Due to the independence of assignment to each parameter, the valuation of a formula's parameters can be decomposed into the valuations of its parameter subset. For instance, the valuation of formula φ_1 can be written as $\theta = [\theta_A, \theta_B, \theta_C, \theta_D]$, where row vectors θ_A, θ_B, θ_C and θ_D denote the valuations of parameter subsets A, B, C and D, respectively.

Reflexivity $\varphi_1 \preceq_P \varphi_1$ is equivalent to $\forall\theta$, $\phi_{1,\theta} \equiv \phi_{2,\theta}$ or $\mathcal{L}(\phi_{1,\theta}) = \mathcal{L}(\phi_{1,\theta})$, which is trivially true.

Transitivity If $\varphi_1 \preceq_P \varphi_2$ and $\varphi_2 \preceq_P \varphi_3$, we have

$$
\begin{aligned}
& \forall\theta = [\theta_A, \theta_B, \theta_C, \theta_D, \theta_E, \theta_F], \phi_{1,\theta} \preceq_S \phi_{2,\theta} \\
& \text{and } \forall\theta' = [\theta'_B, \theta'_C, \theta'_D, \theta'_E, \theta'_F, \theta'_G], \phi_{2,\theta'} \preceq_S \phi_{3,\theta'} \\
\Rightarrow\ & \forall\theta'' = [\theta''_A, \theta''_B, \theta''_C, \theta''_D, \theta''_E, \theta''_F, \theta''_G], \phi_{1,\theta''} \preceq_S \phi_{2,v''} \\
& \text{and } \phi_{2,v''} \preceq_S \phi_{3,\theta''} \\
\Rightarrow\ & \forall\theta'' = [\theta''_A, \theta''_B, \theta''_C, \theta''_D, \theta''_E, \theta''_F, \theta''_G], \phi_{1,\theta''} \preceq_S \phi_{3,\theta''} \\
& \text{due to transitivity of } \preceq_S \\
\Rightarrow\ & \forall\theta''' = [\theta''_A, \theta''_B, \theta''_C, \theta''_D, \theta''_E, \theta''_G], \phi_{1,\theta'''} \preceq_S \phi_{3,\theta'''} \\
\Rightarrow\ & \varphi_1 \preceq_P \varphi_3.
\end{aligned}
$$

Antisymmetry If $\varphi_1 \preceq_P \varphi_2$ and $\varphi_2 \preceq_P \varphi_1$, we have

$$
\begin{aligned}
& \forall\theta, \phi_{1,\theta} \preceq_S \phi_{2,\theta} \text{ and } \phi_{2,\theta} \preceq_S \phi_{1,\theta} \\
\Rightarrow\ & \forall\theta, \phi_{1,\theta} \equiv \phi_{2,\theta} \text{ due to antisymmetry of } \preceq_S \\
\Rightarrow\ & \varphi_1 \equiv \varphi_2
\end{aligned}
$$

□

Further, we have

Lemma 6.1. *The partial order $\preceq_P$ satisfies the following properties.*

1. $\varphi_1 \wedge \varphi_2 \preceq_P \varphi_j \preceq_P \varphi_1 \vee \varphi_2$ *for* $j = 1, 2$

2. $\Box_{[\tau_1,\tau_2)}\ell \preceq_P \Diamond_{[\tau_1,\tau_2)}\ell$, *where* ℓ *is a linear predicate.*

□

The first property is an extension of the propositional logic rules $A \wedge B \Rightarrow A \Rightarrow A \vee B$. The second property states "If a property is always true over a time interval, then it is trivially true at some point in that interval".

DAG and Robustness Degree

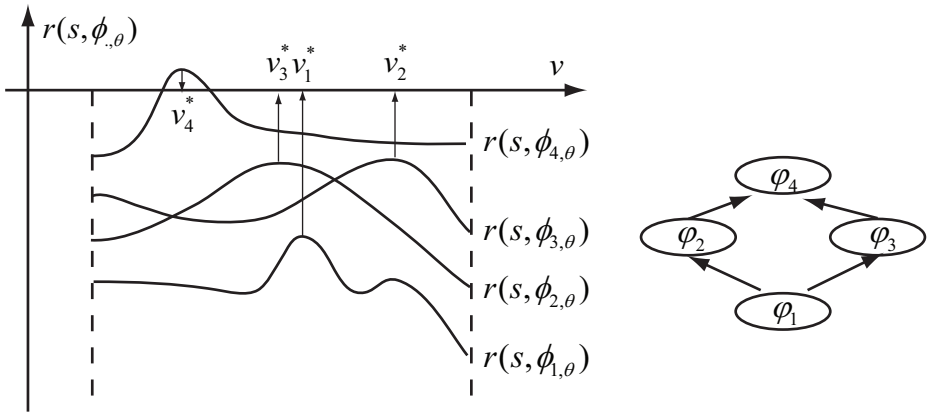


Figure 6·3: Illustration of the relationship between iPSTL formulae and robustness degree.

The structure of iPSTL and the definition of the partial order $\preceq_P$ enable the following theorem.

Theorem 6.1. *The formulae in iPSTL have an equivalent representation as nodes in an infinite DAG. A path exists from formula* φ_1 *to* φ_2 *iff* $\varphi_1 \preceq_P \varphi_2$. *The DAG has a unique top element* ($\top$) *and a unique bottom element* ($\bot$).

Proof. A partially ordered set $< X, \preceq >$ forms a *lattice* if any two elements $x_1, x_2 \in X$ have a join and a meet (Davey and Priestley, 2002). The join and meet can be computed by means of two binary operators, $\sqcup : X \times X \to X$ and $\sqcap : X \times X \to X$, using the supremum and infimum functions, i.e.,

$$\begin{array}{ll} x_1 \sqcup x_2 & := \sup \{x_1, x_2\} \\ x_1 \sqcap x_2 & := \inf \{x_1, x_2\} \end{array} \tag{6.4}$$

Any partially ordered set $< X, \preceq >$ with a lattice structure can be represented by a directed acyclic graph (DAG). First, a Hasse diagram(Davey and Priestley, 2002) can be constructed with each node of the diagram corresponding to an element of X. Then, the DAG can be obtained by adding a direction to each line segment of the Hasse diagram, which point from a 'lower' element (in Cartesian coordinates, has a strictly smaller second coordinate) to a 'higher' element (has a strictly larger second coordinate). The join (meet) of two elements x_1 and x_2 is the 'lowest' ('highest') node where two paths starting from node x_1 and node x_2 and along forward (backward) edges meets.

Proving Theorem 6.1 is equivalent to proving that the set of iPSTL formulae with partial order $\preceq_P$ form a lattice. More formally,

Proposition 6.2. *For all $\varphi_1, \varphi_2 \in \Phi$, their join $\varphi_1 \sqcap \varphi_2$ and meet $\varphi_1 \sqcup \varphi_2$ exist and are unique.*

Proof. Join Treat the subformulae $\Box_I p$ and $\Diamond_I p$ where p is a linear predicate and I is a time interval $I := [\tau_1, \tau_2)$ as different Boolean predicates. Calculate the Disjunctive Normal Form (DNF) of $\varphi_1 \wedge \varphi_2$ (Huth and Ryan, 2004). Then, if $\Box_I p$ and $\Diamond_I p$ coexist in a term replace them with $\Box_I p$; if $\Box_I p$ and $\Diamond_I \neg p$ coexist in a term, we replace them with $\perp$ (False) or equivalently delete the corresponding term;

similarly, if $\Box_I \neg p$ and $\Diamond_I p$ coexist in a term, we delete the corresponding term. The resulting formula is the join $\varphi_1 \sqcap \varphi_2$, which is unique because DNFs are unique.

Meet The existence and uniqueness of $\varphi_1 \sqcup \varphi_2$ can be proved similarly by utilizing the Conjunctive Normal Form (CNF) of $\varphi_1 \vee \varphi_2$. □

Thus, $< iPSTL, \preceq_P >$ is a lattice and therefore has an equivalent representation as an infinite DAG. □

An example of such a DAG is shown in Figure 6·6.

Next, we establish a relationship between the robustness degrees of a signal s with respect to iSTL (iPSTL) formulae ϕ (φ) and the partial order $\preceq_S$ ($\preceq_P$).

Theorem 6.2. *The following statements are equivalent:*

1. $\phi_1 \preceq_S \phi_2$;

2. $\forall s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n), r(s, \phi_1) \leq r(s, \phi_2)$.

Proof. ($\Rightarrow$) Since $\mathcal{L}(\phi_1) \subset \mathcal{L}(\phi_2)$, for any $s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n)$, there are three possibilities: 1) $s \in \mathcal{L}(\phi_1)$; 2) $s \in \mathcal{L}(\phi_2) \cap \mathcal{L}(\neg\phi_1)$; 3) $s \in \mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\neg\phi_2)$. Here, $\mathcal{L}(\neg\phi_1) := \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n) \setminus \mathcal{L}(\phi_1)$ and $\mathcal{L}(\neg\phi_2) := \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n) \setminus \mathcal{L}(\phi_2)$. For Condition 1,

$$\begin{aligned} \mathcal{L}(\phi_1) \subset \mathcal{L}(\phi_2) &\Rightarrow \mathcal{L}(\neg\phi_2) \subset \mathcal{L}(\neg\phi_1) \\ &\Rightarrow \mathcal{L}(\neg\phi_1) = \mathcal{L}(\phi_2) \cup (\mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\phi_2)) \end{aligned}$$

Thus,

$$\begin{aligned} r(s, \phi_1) =& \text{Dist}_\rho(s, \mathcal{L}(\phi_1)) \\ =& \inf\{\rho(s, y) | y \in cl(\mathcal{L}(\neg\phi_1))\} \\ =& \inf\{\rho(s, y) | y \in cl(\mathcal{L}(\phi_2)\end{aligned}$$

$$
\begin{aligned}
&\cup (\mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\phi_2)))\} \\
=&\inf\{\rho(s,y)|y \in cl(\mathcal{L}(\phi_2)) \\
&\text{ or } y \in cl(\mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\phi_2))\} \\
=&\inf\{\inf\{\rho(s,y)|y \in cl(\mathcal{L}(\phi_2))\}, \\
&\inf\{\rho(s,y)|y \in cl(\mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\phi_2))\}\} \\
=&\min\{\inf\{\rho(s,y)|y \in cl(\mathcal{L}(\phi_2))\}, \\
&\inf\{\rho(s,y)|y \in cl(\mathcal{L}(\neg\phi_1) \cap \mathcal{L}(\phi_2))\}\} \\
&\leq \inf\{\rho(s,y)|y \in cl(\mathcal{L}(\phi_2))\} \\
=&\text{Dist}_\rho(s, \mathcal{L}(\phi_2)) \\
=&r(s,\phi_2)
\end{aligned}
$$

Condition 3 can be proved similarly. For Condition 2, since $s \notin \mathcal{L}(\phi_1)$ and $s \in \mathcal{L}(\phi_2)$, we have $r(s,\phi_1) \leq 0$ and $r(s,\phi_2) \geq 0$. Then it is true that $r(s,\phi_1) \leq r(s,\phi_2)$.

($\Leftarrow$) Assume otherwise, then there exists an $s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n)$ such that $r(s,\phi_1) \leq r(s,\phi_2)$ and $s \in \mathcal{L}(\phi_1)$ but $s \notin \mathcal{L}(\phi_2)$. Thus, we have $r(s,\phi_1) \geq 0$ and $r(s,\phi_2) \leq 0$, which results a contradiction, since $r(s,\phi_1)$ and $r(s,\phi_2)$ can not be zero simultaneously. □

Corollary 6.1. *The following statements are equivalent:*

1. $\varphi_1 \preceq_P \varphi_2$;
2. $\forall s \in \mathcal{F}(\mathbb{R}^+, \mathbb{R}^n)$ *and* $\forall \theta, r(s,\phi_{1,\theta}) \leq r(s,\phi_{2,\theta})$.

□

Corollary 6.1 is illustrated in Figure 6·3. The formulae are organized according to the relation $\varphi_1 \preceq_P \varphi_2, \varphi_3 \preceq_P \varphi_4$, which means that $r(s,\phi_{1,\theta}) \leq r(s,\phi_{2,\theta}), r(s,\phi_{3,\theta})$

$\leq r(s, \phi_{4,\theta})$ for all valuations θ.

6.1.3 Supervised Learning Algorithm

In this section, we show how to solve Problem 6.1. The general idea for this procedure is to map the learning problem into an optimization problem by using the robustness degree as an intermediate fitness function. The optimization problem is then solved by combining a discrete search over a DAG to find an iPSTL formula φ with a continuous search to find its appropriate parameterization θ. The final output is an iSTL formula ϕ_θ.

Problem 6.1 can be cast as the following optimization problem.

Problem 6.2. Find an iSTL formula ϕ_{N,θ_N} such that the iPSTL formula φ_N and valuation θ_N minimize

$$J_a(\varphi, \theta) = \frac{1}{L} \sum_{i=1}^{L} l(p_i, r(s_i, \phi_\theta)) + \lambda ||\phi_\theta||, \tag{6.5}$$

where λ is a weighting parameter, and $||\phi_\theta||$ is the length of ϕ_θ (number of linear predicates that appear in ϕ_θ) and l is a loss function, which is chosen to be *hinge loss* in our case

$$l(p_i, r(s_i, \phi_\theta)) = \max(0, \epsilon_r - p_i r(s_i, \phi_\theta)), \tag{6.6}$$

where $\epsilon_r << 1$. □

We continuize l by using the robustness degree as an intermediary fitness function, a measure of how well a given formula fits observed data. Formula length is penalized in our approach because if ϕ_{N,θ_N} grows arbitrarily long, it becomes as complex to represent as the data itself, which would render the inference process redundant.

Theorem 6.1 and Corollary 6.1 have important implications for solving Problem

6.1. The inferred formula should be a close representation of the properties that differentiate normal behaviors from attacked behaviors. Contracting the inferred formula's language by a small amount should result in a formula with a high missed detection rate. Thus, the mined formula should in principle be the lowest ordered formula that satisfies all of the observed normal behaviors. The DAG representation of $iPSTL$ can naturally be used to find such a "barely" satisfying formula. The search starts from the most exclusive formula and follows directed edges until a satisfying formula is found. This is shown in Figure 6·3. A formula is sought to describe the single normal output s. The formulae induced from optimal valuations (denoted with $*$ superscripts) of formulae $\varphi_1, \varphi_2, \varphi_3$ are all still violated by s (have negative robustness degrees). Thus, we have to go up the DAG to formula φ_4 to find a formula that s 'barely' satisfies, i.e. a formula with a small yet positive robustness degree.

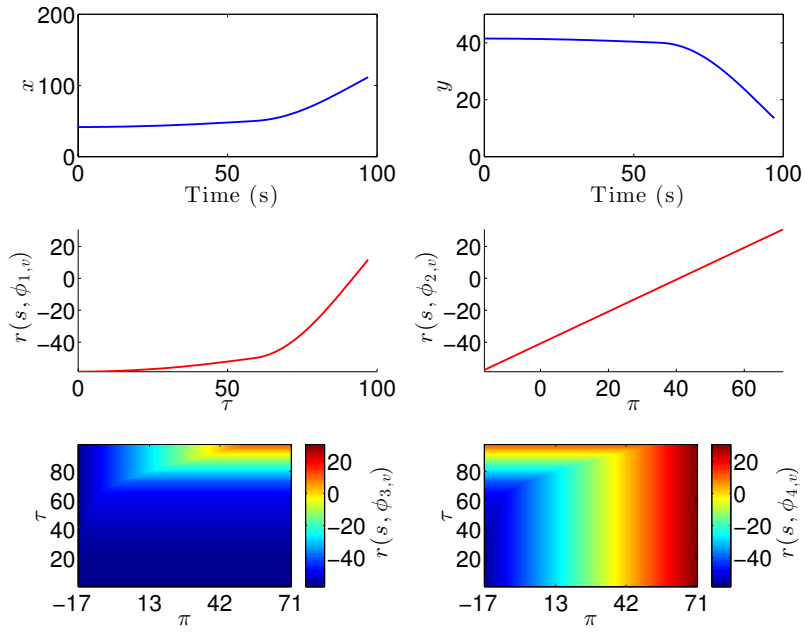


Figure 6·4: Simple example of formula search.

The interaction between the graph search and parameter estimation is further

illustrated in Figure 6·4. Suppose we have a single boat's trajectory s, whose x and y coordinates are shown in the top left and right plots, respectively. The center left (right) figure shows the robustness degree with respect to $\varphi_1 := \diamondsuit_{[0,\tau)}(x > 100)$ ($\varphi_2 := \diamondsuit_{[0,40)}(y < \pi)$) for various values of τ (π). Note that by selecting the parameter $\tau(\pi)$ for each φ_i, we can maximize or minimize the robustness degree of the signal with respect to the induced formula $\phi_{i,\theta}$. The bottom left plot shows the robustness degree for $\varphi_3 := \varphi_1 \wedge \varphi_2$ for various pairs (τ, π) and the bottom right plot shows the robustness degree with respect to $\varphi_4 := \varphi_1 \vee \varphi_2$. Note that $\varphi_3 \preceq_P \varphi_1(\varphi_2) \preceq_P \varphi_4$. By considering φ_3 rather than φ_1 or φ_2 alone, we can find a larger class of iSTL formulae that strongly violate the specification, which is useful for mining formulae with respect to undesirable behavior. Similarly, by considering φ_4, we can find a larger class of formulae that robustly satisfy the behavior. This is useful when we consider large groups of outputs, as it is more likely that for two signals s_1, s_2 where $p_1 = 1, p_2 = -1$, we can find a formula $\phi_{j,\theta}, j \in \{3, 4\}$ such that $r(s_1, \varphi_{j,\theta}) > 0$ and $r(s_2, \varphi_{j,\theta}) < 0$ for $i = 1, 2$ than to be able to find a formula $\phi_{1,\theta}$ or $\phi_{2,\theta}$ that achieves the same classification.

The framework for solving Problem 6.1 is detailed in Alg. 6.1.

Initialization Our algorithm operates on V, the set of all variables represented in the output signals from the system. The inference process begins in line 4 of Alg. 6.1, where DAGInitialization(V) generates the basis of the candidate formulae. The basis is a set of rectangular predicates with temporal operators, called *basis nodes*, of the form $O_{[\tau_1,\tau_2)}(x_s \sim \pi_1)$ where $O \in \{ \square , \diamondsuit \}$, $\sim \in \{\geq, <\}$ and $x_s \in V$. That is, the basis represents all internal formulae φ_i of length 1. Edges are constructed from φ_j to φ_k in the initial graph $\mathcal{G}_1$ iff $\varphi_j \preceq_P \varphi_k$. For example, in the naval surveillance example if we only consider the (x, y) position of the boat, then the initial graph is shown in Figure 6·5.

Algorithm 6.1 Unsupervised Learning

```
1: function UnsupervisedLearning(\{(s_i, p_i)\}_{i=1}^{M}, V, \delta, L_{max})
2: for i = 1 to L_{max} do
3:    if i = 1 then
4:       \mathcal{G}_1 \leftarrow DAGInitialization(V);
5:       List \leftarrow ListInitialization(\mathcal{G}_1);
6:    else
7:       \mathcal{G}_i \leftarrow PruningAndGrowing(\mathcal{G}_{i-1});
8:       List \leftarrow Ranking(\mathcal{G}_i \setminus \mathcal{G}_{i-1}, );
9:    while List \neq \emptyset do
10:      \varphi \leftarrow List.pop();
11:      (\theta, cost) \leftarrow ParameterEstimation(\{(s_i, p_i)\}_{i=1}^{M}, \varphi)
12:      if cost \leq \delta then
13:         return (\varphi, \theta).
14: return MinimumCostNode(\mathcal{G}_{L_{max}});
```

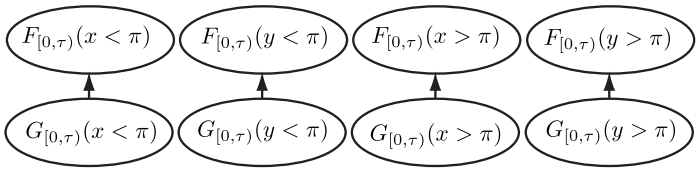


Figure 6·5: The initial graph $\mathcal{G}_1$ constructed from x, y coordinates.

ListInitialization($\mathcal{G}_1$) (line 5) generates a ranked list of formulae from the basis nodes. Since we do not yet know anything about how well each of the basis nodes classifies behaviors, the rank (used in parameter estimation) is generated randomly.

Parameter Estimation After the graph is constructed, we find the optimal parameters for each of the nodes. The candidate formulae in *List* are iterated through from lowest rank to highest (line 10). ParameterEstimation($\{(s_i, p_i)\}_{i=1}^{L}, \varphi$)(line 11) uses simulated annealing (Russell and Norvig, 1995) to find an optimal valuation for φ by minimizing the cost J_{ai}.

Structural Inference After the first set of parameters and costs have been found, the iterative process begins. The definition of the partial order allows for dynamic extension of the formula search space. We cannot explicitly represent the infinite

DAG, so we construct a finite subgraph of possible candidate formulae and expand it when the candidate formulae perform insufficiently. PruningAndGrowing($\mathcal{G}_{i-1}$) (line 7) does this by first eliminating a fixed number of nodes with high costs, i.e. those formulae that do not fit the observed data. Pruning the graph to eliminate high cost formulae follows naturally from forward subset selection ideas developed in machine learning (Trevor et al., 2001). Then, the function grows the pruned $\mathcal{G}_{i-1}$ to include nodes with length i according to graph manipulation rules detailed in Section 6.1.2. An example of a subset of a graph $\mathcal{G}_2$ grown from the (pruned) basis graph is given in Figure 6·6.

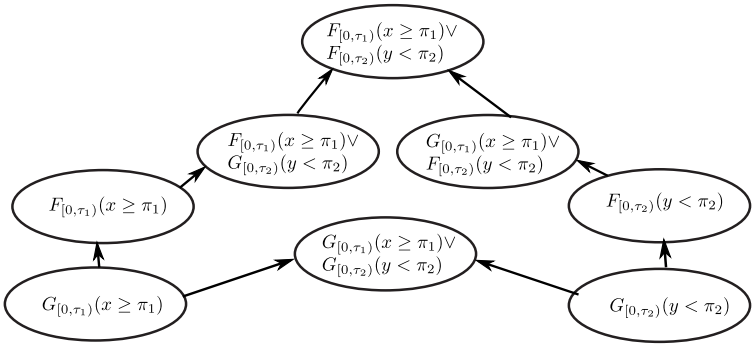


Figure 6·6: A subset of the DAG $\mathcal{G}_2$ after pruning and expansion. For compact representation, only the internal formulae φ_i are shown.

Ranking($\mathcal{G}_i \setminus \mathcal{G}_{i-1}$) (line 8) ranks the newly grown nodes based on a heuristic function

$$\frac{1}{|pa(k_i)|} \sum_{k_{i-1} \in pa(k_i)} J_a(k_{i-1}), \tag{6.7}$$

where k_i is a node in $\mathcal{G}_{i-1}$, $pa(k_i)$ is the set of k_i's parents, and $|pa(k_i)|$ is the size of $pa(k_i)$. For example, in Figure 6·6, for $k_i = (\ \diamondsuit_{[0,\tau_1)}(x \geq \pi_1) \wedge (\ \diamondsuit_{[0,\tau_2)}(y < \pi_2))$, $pa(k_i) = \{\ \diamondsuit_{[0,\tau)}(x \geq \pi), (\ \diamondsuit_{[0,\tau)}(y < \pi)\}$ and $|pa(k_i)| = 2$.

The iterative graph growing and parameter estimation procedure is performed until a formula with low enough misclassification rate is found or L_{max} iterations are

completed. At this point, MinimumCostNode($\mathcal{G}_i$) returns the node with the minimum cost within $\mathcal{G}_i$.

Complexity Without pruning, the discrete layer of the described algorithm runs in time $\mathcal{O}(L_{max} \cdot 2^{|V|})$. Since PruningAndGrowing prunes a constant number of nodes at each iteration, the complexity of the discrete layer is reduced to $\mathcal{O}(L_{max} \cdot |V|^2)$ when pruning is applied. The continuous layer of the algorithm, based on the simulated annealing algorithm, runs in time $\mathcal{O}(L_{max}(n^2 + m) \cdot \log(L))$, with n being the number of samples used in simulated annealing, and m being the number of data points per signal.

Remark 6.1. It has been shown in (Fainekos, 2011) that the set of all linear temporal logic (LTL) formulae can also be organized in a DAG using a partial order similar to $\preceq_S$. However, unlike LTL, PSTL can express temporal specifications involving continuous-time intervals and constraints on continuously valued variables. To our own knowledge, our algorithm is the first of its kind which can be used to infer an STL formula by inferring both its PSTL structure and its optimal valuation. □

6.1.4 Case Studies

Algorithm 6.1 was implemented in a software tool called TempLogIn (TEMPoral LOGic INference) in MATLAB. We developed all of the components of our solution in-house, including the graph construction, search algorithms, and the simulated annealing algorithm. The software is available at `http://hyness.bu.edu/Software.html`.

Synthesized data

In this case study, we apply our supervised learning algorithm to the naval surveillance scenario used in Example 6.1. We model each vessel as a Dubins' vehicle:

$$\begin{cases} \dot{x} &= v\cos\alpha \\ \dot{y} &= v\sin\alpha \\ \dot{\alpha} &= \omega, \end{cases} \tag{6.8}$$

where x and y are the vessel's coordinates, v is its constant speed, α is its heading, and ω is its angular velocity. Further, assume that the x and y coordinates collected by the AIS are subjected to an additive white Gaussian noise $\mathcal{N}(0, 0.1)$.

We generated 50 trajectories demonstrating normal behaviors, 25 trajectories demonstrating suspicious behaviors consistent with human trafficking behaviors, and 25 trajectories demonstrating suspicious behaviors consistent with terroristic behaviors. A subset of these trajectories are shown in Figure 6·10. Our goal was to find a formula that described only the normal behaviors from this training set. Using our implementation of Algorithm 6.1 with $n = 15$, $m = 15$ yielded the formula

$$\phi_N = \quad \diamondsuit_{[0,229)}(\ \square_{[28,227)} y_s > 21.73) \wedge (\ \square_{[308,313)} x_s < 34.51) \tag{6.9}$$

with total misclassification rate 0.0950. The total computation time was 1313 s (approx. 22 minutes) on a computer with 2.41 GHz processor and 7.4 GB RAM.

In plain English, this formula reads "Within 229 minutes, the boat's y coordinate remains less than 21.73 dam and dips below 34.51 dam". The blue lines in Figure 6·10 correspond to the thresholds in (6.9).

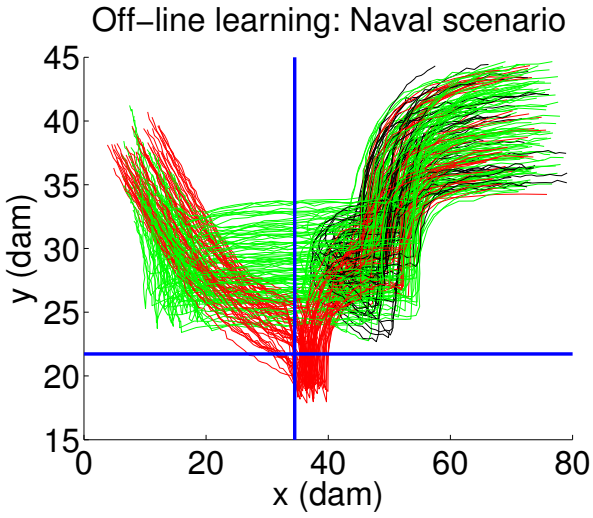


Figure 6·7: Results of offline inference. The green trajectories represent normal behaviors, the red trajectories represent human trafficking, and the black trajectories represent terroristic activity. The blue lines are boundaries given by the formula (6.9)

Experimentally derived data

In addition to the naval surveillance example, we consider a more realistic example from synthetic biology. In this field, gene networks are engineered to achieve specific functions (Kirby, 2010; Salis et al., 2009). The robustness degree has previously been exploited in gene network design and analysis (Rizk et al., 2008; Bartocci et al., 2013; Donzé et al., 2011), but to our knowledge, specification mining has never been used to analyze a gene network. We consider the gene network presented in (Gol et al., 2013). The network, shown in Figure 6·8, controls the production of two proteins, namely $tetR$ and RFP. This network is expected to work as an inverter in which the concentrations of $tetR$ and RFP can be treated as the input and the output, respectively. In particular, $tetR$ represses the production of RFP. A high $tetR$ concentration decreases the production rate of RFP, hence the concentration of RFP eventually decreases and stays low. Similarly, if the concentration of $tetR$ is low, then the production of RFP is not repressed, and its concentration eventually

increases and stays high.

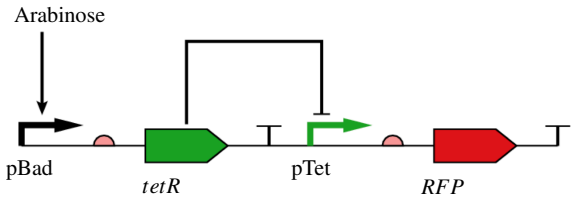


Figure 6·8: A synthetic gene network. The genes coding for proteins $tetR$ and RFP are shown as colored polygons. The promoters (pBad and pTet) regulating protein production rates are indicated by bent arrows. The regulators (*arabinose* and $tetR$) are connected to the corresponding promoters.

In (Gol et al., 2013), a stochastic hybrid system modeling the gene network was constructed from characterization data of the biological network components. Statistical model checking was used to check a temporal logic formula (expressing a property chosen by biology experts) that describes the inverter behavior of the network. In this paper, sample trajectories of this system [1] are used to find a formula that describes the inverter behavior without any prior expert knowledge.

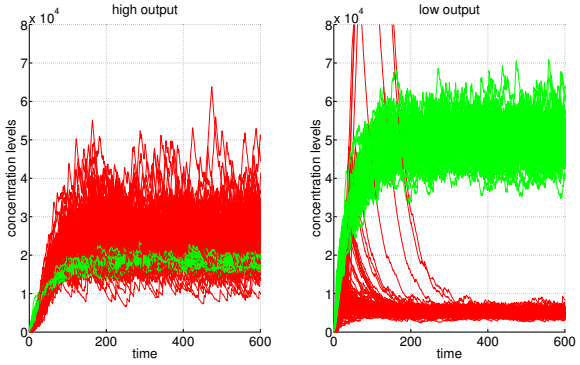


Figure 6·9: Concentration levels x_{tetR} (green) and x_{RFP} (red) for the high and low output cases. 100 signals are plotted for each protein and each case.

We generated 600 signals, half of which correspond to the low output case (repression) and half of which correspond to the high output case (no repression). Figure 6·9

[1] *Arabinose* regulates the production rate of $tetR$. We use trajectories generated at different concentration levels of *arabinose*. As we are interested in cause-effect relationship between $tetR$ and RFP, we omit the concentration of *arabinose*.

shows 200 of these signals. Assume that we are interested in characterizing the low output case ($p_i = 1$ for low output signals). The inferred rSTL formula ϕ_{des} which classifies both cases and describes the pre-conditions for low output is

$$\begin{aligned}\phi_{des} = \quad & \diamondsuit_{[0,118)}(\square_{[0,340)}(x_{tetR} \geq 23209) \Rightarrow \\ & \square_{[188,323)}(x_{RFP} < 13479))\end{aligned} \tag{6.10}$$

This formula captures the repressing effect of $tetR$, and shows that the designed gene network works as expected. In particular, the formula implies that $tetR$ represses the production of RFP when its concentration is higher than 23209 for 340 time units. Moreover, when the production of RFP is repressed, its concentration drops below 13479 within 188 time units. Such quantitative information learned from the formula helps the user to design more complex gene networks.

On the same computer used in the first case study, the inference procedure took a total of 1188 seconds. [2] For the biological network case study, the number of samples generated by the simulating annealing, n, and the maximum number of data points per signal, m, were 100 and 600, respectively.

6.2 On-line Supervised Learning of STL Formulae

In this section, we present an alternative approach to solve the supervised learning problem (Problem 6.1) considered in Section 6.1. The goal of the problem remains the same, namely to infer an iSTL formula that is used to differentiate desirable behaviors from undesirable behaviors, In the on-line learning case, however, instead of having an entire database of labeled trajectories *a priori*, the learning agent receives labeled

[2] This problem was solved using rPSTL instead of iPSTL (Kong et al., 2014). This is a minor restriction that leads to a slight difference in the algorithm used to solve it. When rPSTL is used, we first apply the inference procedure to data after time t_{trunc}, learn a formula that classifies this data, and then use this information to guide the search for the entire formula when running the inference algorithm over all the data.

trajectories sequentially. Our approach is based on stochastic gradient descent with numerical rather than analytic derivatives. In addition to being more computationally efficient than the procedure given in Section , the nature of sequential learning makes the on-line approach more appropriate for systems in which no or very little historical data is available.

6.2.1 Problem Definition

On-line supervised learning is defined by Problem 6.3.

Problem 6.3. A system or a group of systems produce outputs s_i with expert-given labels p_i as defined in Problem 6.1. Maintain a formula ϕ_N^t such that the misclassification rate $MR(\{(s_i, p_i)\}_{i=1}^t, \phi_N^t)$ as defined in Problem 6.1 is minimized. When a new pair (s_{t+1}, p_{t+1}) becomes available, use ϕ_N^t and the new pair to construct ϕ_N^{t+1}. □

Problem 6.3 was addressed in (Kong et al., 2015). An on-line version of the anomaly detection problem can be similarly proposed. However, due to the inherent difficulty in on-line unsupervised learning, this remains an open problem.

6.2.2 On-line Supervised Learning Algorithm

Here, we consider how to extend Alg. 6.1 to solve Problem 6.3. In principle, optimization problems such as (6.5) can be solved for on-line settings via stochastic gradient descent (Trevor et al., 2001; Kivinen et al., 2004). With mild assumptions, for a fixed iPSTL formula structure φ, such a method can find its optimal parameterization θ^* if there exists a ϕ_{θ^*} with structure φ that can classify the data. Let θ_i be the parameterization of φ after i pairs of signals and labels have been observed.

Algorithm 6.2 Online Learning

```
1: function OnlineLearning({(s_i, p_i)}_{i=1}^L, η_max, η_min, α, checkInt, numIter)
2: for φ_k ∈ formulae do
3:    θ^k = ParameterEstimation((s_i, p_i), φ_k);
4:    for i = 1, ... numIters do
5:       traces= UpdateTraces(traces, s_i, p_i);
6:       for (φ_k, θ^k) in formulae do
7:          θ^k = ParameterUpdate((s_i, p_i), φ_k, θ^k);
8:       η = max(αη, η_min)
9:       if i mod checkInt == 0 then
10:         for (φ_m, θ^m) in do
11:            formulae ← ReplaceFormulae(formulae)
12:         η ⇒ η_max
13: return BestFormula(formulae);
```

The stochastic gradient descent that minimizes the loss function l is given by

$$\theta_{i+1} = \begin{cases} \theta_i & \text{if } p_i r(s_i, \phi_{\theta_i}) \geq \epsilon_r, \\ \theta_i + \eta \frac{\partial r}{\partial \theta} p_i & \text{otherwise.} \end{cases} \tag{6.11}$$

where $\eta > 0$ is a learning rate and the partial derivative is calculated according to the centered first distance. If φ is the correct iPSTL formula for classification, it should be expected that there exist a step $\bar{i}$ such that $p_i r(s_i, \phi_{\theta_i}) \geq 0$ for all $i \geq \bar{i}$. That is to say the total misclassification rate approaches 0. [3] If, on the other hand, φ is not the correct formula, then the improvement on classification performance saturates at a certain step $\tilde{i}$ and a new formula should be sought.

We propose an on-line learning procedure described by Alg. 6.2. This new procedure can learn a formula ϕ_θ as the labeled signals (s_i, p_i) arriving sequentially. This procedure operates on a collection of N_f candidate formulae $\{\varphi_j, \theta^j\}_{j=1}^{N_f}$. The algorithm operates on this collection instead of considering a single formula at a time

[3] The difference between subsequent valuations θ_i and θ_{i+1} should not be too extreme, as this would cause the valuation to oscillate about the optimal value θ^*. Therefore, the learning rate η should be chosen to be a small, positive number or to decrease with respect to the iteration number.

because there is initially very little information about what kinds of behaviors the system may satisfy. This is still more computationally efficient than the offline learning method, however, because for each trajectory and label pair (s_i, p_i) are introduced to the algorithm, N_f robustness calculations are performed, in contrast to the n^2m calculations that are performed by the simulated annealing algorithm.

The algorithm initializes the set of formulae to N_f formulae from the basis and corresponding initial valuation guesses that are found via simulated annealing with small values of n, m. Then, when a new trajectory and label pair (s_i, p_i) becomes available, the function ParameterUpdate is called to update each value θ^j in the formula database according to the rule (6.11). Every checkInt trajectories, the misclassification rates of each φ_{j,θ^j} with respect to the trace database are evaluated and the formula database is then populated with new formulae.

The replacement formulae are constructed from the best performing formulae according to Algorithm 6.3. Pairs of the best performing formulae $(\phi_{b,\theta^b}, \phi_{sb,\theta^{sb}})$ are selected from the best-performing formulae. If the missed detection rates are greater than the false alarm rates, e.g. the size of the languages of the formulae are too large, then the conjunction of the two formulae is added to the formula database. Otherwise, the disjunction of the two is added. The subroutine simplify removes tautologies from the constructed formula. The subroutine getValuation maps the two valuations θ^b, θ^{sb} to the corresponding valuation θ_{new} of the simplified combined formula.

Our algorithm uses a variable learning rate η throughout the on-line inference procedure. We initially start at a high rate η_{max} and decrease it in a geometric fashion with rate $0 << \alpha < 1$ until it reaches a level η_{min}. Whenever the formula database is replaced, this rate is reset to its original level. The variable learning rate allows the ParameterUpdate formula to make bigger steps whenever we initially know very little about the optimal parameterizations for each structure and make smaller,

finer steps after more information has been collected.

Algorithm 6.3 UpdateFormulae. Inputs are a database of traces tr and a database of formulae f.

```
1: function UpdateFormulae(tr,f)
2: for k = 1 to N_f do
3:    (φ_b, θ^b) = bestFormula(f,tr)
4:    uf1 = uf1 ∪ {(φ_b, θ^b)}
5:    (mdb,fab) = calculateRates(φ_b, θ^b, tr)
6:    for m = 1 to N_f − k do
7:       (φ_sb, θ^sb) = bestFormula(f \ (uf1 ∪ uf2), tr)
8:       (mdsb,fasb) = calculateRates(φ_sb, θ^sb, tr)
9:       if (mdsb+mdb > fab+fasb) then
10:         φ_new = simplify(φ_b ∧ φ_sb)
11:      else
12:         φ_new = simplify(φ_b ∨ φ_sb)
13:      θ_new = getValuation(φ_new, θ^b, θ^sb)
14:      f = f ∪{(φ_new, θ_new)}
15:      uf2 = uf2 ∪(φ_sb, θ_sb)
16: return f
```

Complexity As mentioned above, each time a new trajectory and label pair is introduced to the online learning procedure, $\mathcal{O}(N_f)$ robustness calculations with complexity $\mathcal{O}(|\varphi_j|)$ are performed. If N_r formula database updates are performed, the worst-case formula length is 2^{N_r-1}, though this represents an unlikely extreme situation in which no tautologies are introduced and the longest formulae in the database are always among the best-performing. The user has some control over the maximum length of the formula via the parameter checkInt which determines how often the formula replacement occurs. In practice, the maximum formula length is determined by the length of the shortest formula that can separate the two classes of trajectories.

6.2.3 Case Study

We used a larger set of signals of the naval scenario and inferred a formula by using our on-line learning algorithm. At each iteration of our algorithm, we drew a signal

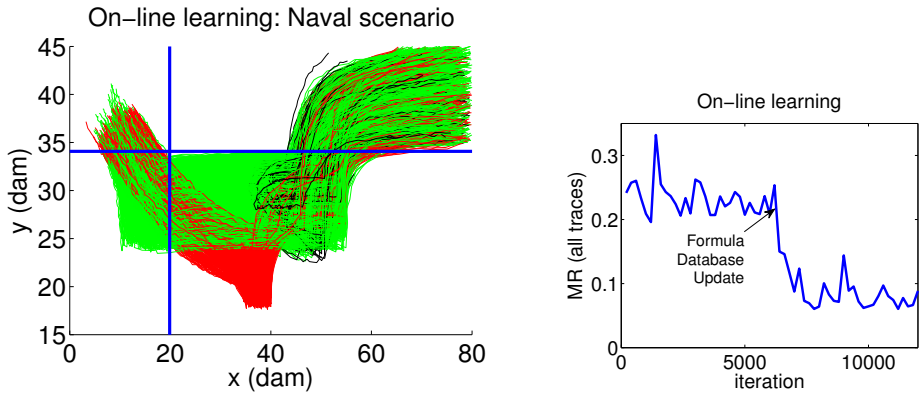
uniformly at random from a set of 2000 trajectories, which consisted of 1000 normal trajectories, 500 human trafficking trajectories, and 500 terrorist trajectories. The learning rate parameters we used were $\alpha = 0.995, \eta_{max} = 0.2$, and $\eta_{min} = 0.01$. Figure 6·10(b) shows the misclassification rate of the inferred formula at time i with respect to all 2000 traces. As we can see, the rate does not monotonically decrease, but it does decrease to a point. The formula that was inferred after all 2000 trajectories were used is

$$\begin{aligned} \phi_N = \quad & \diamondsuit_{[0,229)}(\square_{[174,228)} x_s < 19.88) \\ & \wedge(\square_{[92,297)} y_s < 34.08) \end{aligned} \tag{6.12}$$

In plain English, this means that "At some point within 229 minutes, the boat's x coordinate remains below 19.88 between 174 and 228 minutes in the future and its y coordinate remains less than 34.08 dam between 92 and 297 minutes in the future." The space parameters for (6.12) are illustrated in Figure 6·10(a). The total misclassification rate of the final formula was 0.0885. The total computation time was 996 s (approx. 16 minutes) on a computer with 2.41 GHz processor and 7.4 GB RAM. This computation time included evaluating misclassification rates for the entire set at certain time intervals, i.e. generating Figure 6·10(b). Using the same parameters and not doing this calculation yields a computation time of 648 s (approx. 11 minutes). This represents a significant computational speedup over the off-line method, especially when we consider that the method operated on a larger data set.

6.3 Unsupervised Learning of STL Formulae

In this section, we extended the supervised learning machinery we presented in Section 6.1 (Kong et al., 2014) to a model-agnostic unsupervised learning algorithm. This technique infers an STL formula from unlabeled system output data that can be used



(a) (b)

Figure 6·10: (a) Results of on-line inference. The green trajectories represent normal behaviors, the red trajectories represent human trafficking, and the black trajectories represent terroristic activity. The blue lines are boundaries given by the formula (6.12). (b)The misclassification rate over time for on-line learning with respect to all traces.

to classify data as normal or anomalous.

6.3.1 Problem Definition

Here, we consider a more challenging version of Problem 6.1 in which the inference procedure learns from historical data without the knowledge of whether a given output was produced by a system behaving normally or abnormally. More formally, we wish to solve Problem 6.4.

Problem 6.4. From the set $\{s_i\}_{i=1}^M$ (defined in Problem 6.1), find an an iSTL formula ϕ_N such that the misclassification rate

$$MR_D(\{s_i\}_{i=1}^M, \phi_N) = \frac{FA_D + MD_D}{M}$$

is minimized, where

$$FA_D = |\{s_i | s_i \not\models \phi, x_i \text{ is normal } \}|$$

and

$$MD_D = |\{s_i | s_i \models \phi, x_i \text{ is anomalous } \}|.$$

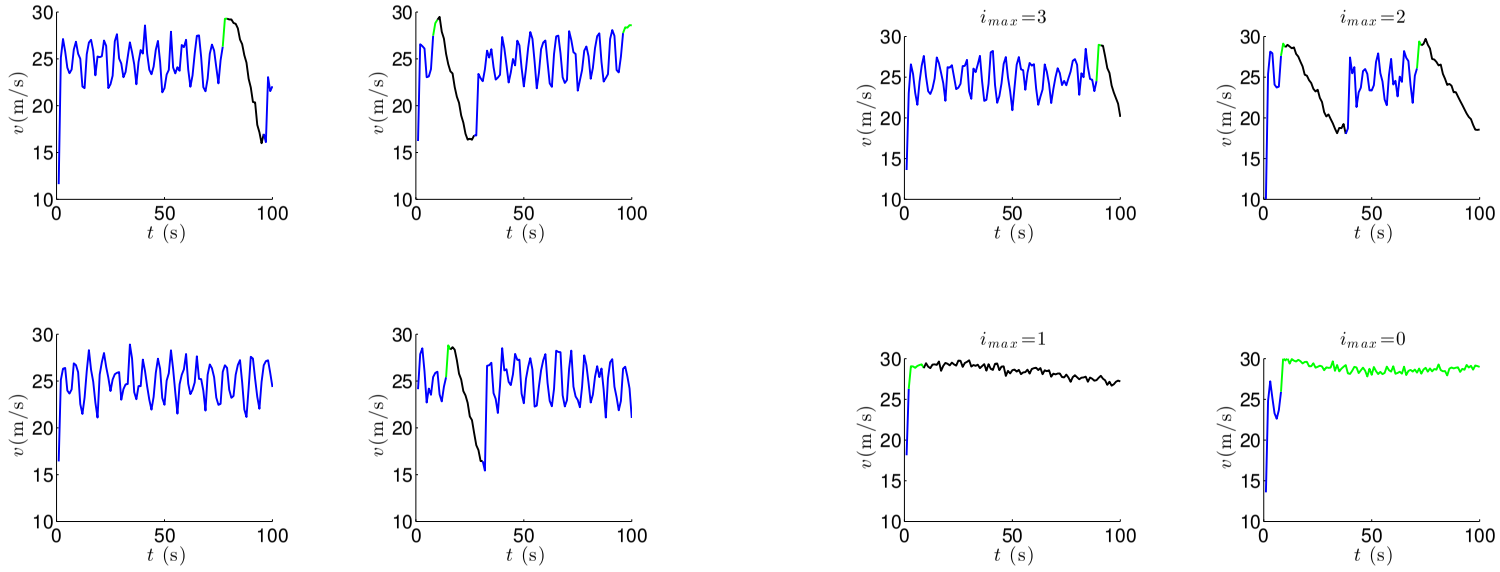
□

Here, we give a motivating example that illustrates Problem 6.4.

Example 6.2 (Train Network Monitoring)**.** Consider a train using an electronically-controlled pneumatic (ECP) braking system. The train has 3 cars, each of which has its own braking system. Our model of the train system is modified from (Sistla et al., 2011). See Section 6.3.3 for more details.

Normal Behavior In this model, the braking system is automated to regulate the velocity v below unsafe speeds and above low speeds to ensure that the train reaches its destination. When the train begins moving, its brakes are inactive and its speed oscillates noisily about a nominal value (25 m/s). However, as a consequence of the system dynamics, if $v(t)$ exceeds a threshold v_{max} (28.5 m/s), the velocity of the un-braked system does not decrease (shown in green in Figure 6·11(a)). Each of the brakes responds to the threshold crossing after a random time delay by engaging. The brakes decrease the velocity of the train (shown in black in Figure 6·11(a)) until it passes a second threshold v_{min} (20 m/s). After random delays, the brakes disengage and the speed resumes oscillation.

Anomalous Behavior An adversary has the possibility to disable each of the brakes of the train. A few sample outputs from the attacked system are shown in Figure 6·11(b). The behavior of the system depends on how much access the adversary has to disrupt its operation. Let b be the number of brakes the agent can affect. If the adversary can only disable one brake ($b = 1$), more time is required for the velocity to be decreased after v_{max} is exceeded, but the velocity is still regulated. If the adversary can disable two brakes ($b = 2$), then the deceleration is further slowed and the train



(a) (b)

Figure 6·11: (a)Output signals of the velocity of the train controlled by three ECP braking systems when all three brakes are functioning normally. The colors refer to modes of the hybrid automaton that describes the system (given in Section 6.3.3): blue indicates the system oscillates between low and high velocities, green indicates the system is moving too fast, and black indicates the system is being braked. (b)Outputs of the train velocity system under different attack scenarios. An adversary has the ability to disable one, two, or three of the trains brakes in order to deregulate its velocity. The variable b is the number of brakes affected by attack.

spends an unacceptable amount of time in its upper operating region. Finally, if the adversary can affect all three brakes, then braking can't happen ($b = 3$), meaning the train will travel indefinitely. All three attacked outputs behave qualitatively differently from each other, but they all clearly violate the desired invariant behavior. □

In order to simplify Problem 6.4 we make two key assumptions. First, anomalous behavior happens infrequently. That is, given a system output s_i, the *a priori* probability that it is anomalous is low. This allows us to classify as anomalous those signals that behave differently from the majority. Second, the outputs of a system behaving abnormally differ qualitatively from the outputs of a system behaving normally. Otherwise, it is impossible to infer any classifier to separate the two sets of outputs. Both assumptions are plausible for real-world scenarios and are commonly made in other anomaly detection problems (Chandola et al., 2009).

6.3.2 Unsupervised Learning Algorithm

Since Problem 6.4 is an unsupervised learning problem, we use some notions from classical unsupervised learning to aid in our approach. In particular, we consider one-class support vector machines (SVMs). A one-class SVM is an optimization technique that, given a set of data, lifts the data to a higher-dimensional feature space and constructs a surface in this space that separates normal data from anomalous data (Shin et al., 2005). Since we use a different paradigm and do not need to lift the problem to a feature space, we ignore many of the feature-specific components of the problem and adapt the objective function used in one-class SVMs to our problem. Thus, we map Problem 6.4 to the following optimization.

Problem 6.5. Find an iPSTL formula ϕ_{N,θ_N} such that the formula φ_N and valuation

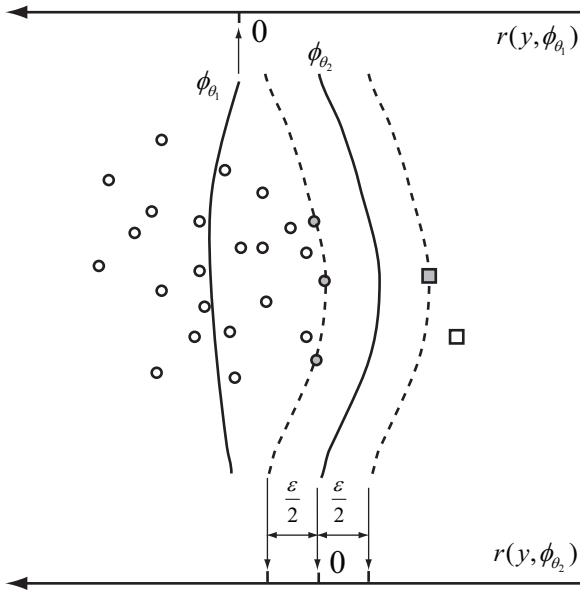


Figure 6·12: A conceptual illustration of our anomaly detection algorithm. Normal outputs are denoted by circles and anomalous ones are denoted by squares.

θ_N minimize

$$\min_{\phi_\theta,\epsilon} d(\phi_\theta) + \frac{1}{\nu N}\sum_{i=1}^{M}\mu_i - \epsilon \tag{6.13}$$

such that

$$\mu_i = \begin{cases} 0 & r(s_i,\phi_\theta) > \frac{\epsilon}{2} \quad \forall i, \\ \frac{\epsilon}{2} - r(s_i,\phi_\theta) & \text{else} \end{cases} \tag{6.14}$$

where ϕ_θ is an iSTL formula, ϵ is the "gap" in signal space between outputs identified as normal and outputs defined as anomalous, ν is the upper bound of the *a priori* probability that a signal x_i is anomalous (Shin et al., 2005), and μ is a slack variable. μ_i is positive if s_i does not satisfy ϕ_θ with minimum robustness $\frac{\epsilon}{2}$. The function d is a "tightness" function that penalizes the size of $\mathcal{L}(\phi_\theta)$. □

By minimizing the sum of the μ_i, optimization (6.13) minimizes the number of traces the learned formula ϕ_θ classifies as anomalous. This is consistent with our assumptions on the problem. By maximizing the gap ϵ, optimization (6.13) attempts to maximize the separation between normal and anomalous outputs. By minimizing

the function $d(\phi_\theta)$, optimization (6.13) prevents the learned formula from trivially describing all observed signals (i.e. finding a formula such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{S})$), which would render the optimization redundant.

Figure 6·12 illustrates how the optimization formulation (6.13) can be used to solve the anomaly detection problem. For the sake of simplicity, let's assume the iPSTL formula $\varphi_{(.)}$ is fixed and we need to infer its optimal valuation $\theta \in \mathbb{R}$. We further assume that $r(s_i, \phi_\theta)$ is monotonically increasing with respect to θ, i.e., for $\theta_1 < \theta_2$ and any signal s_i, $r(s_i, \phi_{\theta_1}) < r(s_i, \phi_{\theta_2})$. In such a case, we can choose $d(\phi_\theta) = \theta$. With $\theta = \theta_1$, a significant number of μ_i are still non-zero, so a larger θ such as θ_2 is needed. Similar to the anomaly learning case, solving (6.13) requires searching over the set of continuous variables (θ and ϵ) as well as over the discrete set of iPSTL formula structures (the structure φ of ϕ_θ). Alg. 6.1 can be adapted to solve (6.13) with the following two changes:

1. The input signals are not labeled, i.e., the inputs are $\{s_i\}_{i=1}^{M}$.

2. ParameterEstimation solves (6.13) instead of (6.5).

Algorithm 6.4 Tightness Function

function Tightness(θ,φ)
$k = 0$
for all parameters $(\tau_1, \tau_2, \pi) \in \theta$ such that $O_{[\tau_1,\tau_2)} x_i \sim \pi)$ in φ **do**
    tightness[k] = Normalize(τ_1); $k++$;
    if $\sim$ is $<$ **then**
        tightness[k] = Normalize(π); $k++$;
    else
        tightness[k] = 1-Normalize(π); $k++$;
return $\lambda \frac{D + \sum_k \text{tightness}[k]}{\text{length(tightness)} + ||\varphi||}$

The ParameterEstimation procedure uses the heuristic tightness function d when calculating the objective function in (6.13). In this paper, we use a heuristic that

penalizes the size of τ_1, as as for monitoring purposes we would prefer to infer formulae that can describe behaviors of the early parts of the system's outputs. For each predicate appearing in ϕ, if the comparison operator is $<$, the size of π is penalized, as the size of the language of $(x_s < \pi)$ increases with π. If the comparison operator is $>$, small values of π are penalized for the same reason. Please see (Jones et al., 2014) for more details. The time complexity of the anomaly detection algorithm is the same as that of the anomaly learning algorithm.

6.3.3 Case Study

In this subsection, we apply our anomaly detection algorithm to the train braking scenario used in Example 6.2. We model the train as a classical hybrid automaton, whose definition is given in (Lygeros et al., 1999). In this example, an adversary can disable the brakes of the system and cause its velocity to become unregulated. More details on the particular model we used can be found in (Sistla et al., 2011; Jones et al., 2014).

A hybrid automaton produces trajectories $x : \mathbb{R}^+ \to X$, where $X \subseteq \mathbb{R}^n$ is a continuous state space. The dynamics of a trajectory x depend on the current discrete mode $q \in Q$ (denoted by a vertex of a graph) of the automaton. The mode of the system changes (denoted by edges of a graph) if a *guard condition* over the state of the system is satisfied. If a transition occurs, the state of the system may change discontinuously according to a *reset relation*. Here we denote guards in black text and reset relations in red text over transition edges in automata.

The hybrid automaton H which describes the total model of the train consists of 3 identical braking subsystems with modes $Q_{b_k} = \{q_{b_k,j}\}_{j=1}^{5}$ that describe the state of each brake and a velocity subsystem with modes $Q_v = \{q_{v,j}\}_{j=1}^{3}$ which describes the dynamics of the train's velocity. The mode of the system is a tuple $q \in Q = Q_v \times$

$\prod_{k=1}^{3} Q_{b_k}$. We do not explicitly show the entire system. The interdependence of the velocity and brake subsystems are given by the dynamics of the velocity subsystem, guard conditions, and reset relations.

The subsystem associated with brake 1 is shown in Figure 6·13(a). The noise processes $n_1 \ldots n_5$ are all Gaussian processes with variance 1, 0.1, 0.3, 3, and 3, respectively. The brake remains in mode $q_{b_1,1}$ during acceleration until $v(t)$ exceeds a threshold v_{max}. At this point, the timer c_1 is initialized to a random value generated by the process n_1, which models the delay between velocity exceeding v_{max} and the engagement of the brake. In the absence of attack, the system stays in the delayed mode $q_{b_1,2}$ until the delay is complete, at which point the brake engages (mode $q_{b_1,3}$). After the velocity is decreased below v_{min}, the system transitions to a second mode $q_{b_1,4}$ in which the brake is still engaged. The counter c_2 models the delay between the speed decrease and brake disengagement. Once the brakes are disengaged, the system returns to mode $q_{b_1,1}$.

The velocity subsystem is shown in Figure 6·13(b). The velocity of the train begins in mode $q_{v,1}$ (blue in Figures 6·11(a) and 6·11(b)) and accelerates to v_{max}. At this point, the dynamics of the train shift to the higher velocity mode $q_{v,2}$ (green in Figures 6·11(a) and refnormalTraces(b)). Once at least one brake engages, the system transitions to a decelerating mode (black in Figures 6·11(a) and refnormalTraces(b)). Once the brakes have all disengaged, the train returns to mode $q_{v,1}$.

An adversary can disable each of the brakes. If an adversary attacks the first brake (denoted by the exogenous event $attack_1$ in magenta), the brake system transitions to a mode $q_{b_1,5}$. The brake cannot transition out of this mode to reach mode $q_{b_1,3}$, so the brake will never be engaged. The events $attack_2$ and $attack_3$ similarly inactivate the second and third brakes, respectively.

We used the train model to generate 50 outputs of H. 43 of the trajectories were

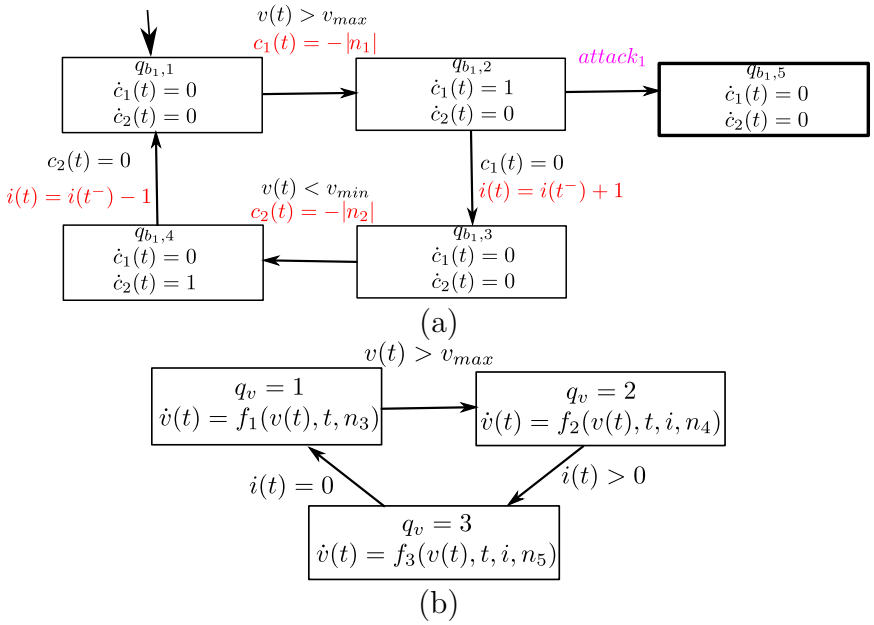


Figure 6·13: (a) ECP braking subsystem of the first car in the train. (b) Velocity subsystem of the entire train.

from normal operation and 7 were from an attacked operation. We only considered attacks in which all of the brakes were disabled ($b = 3$). Our algorithm inferred the formula

$$\begin{aligned}\phi = \quad & \Diamond_{[0,100)}(\Diamond_{[10,69)}(y < 24.9) \\ & \wedge \Diamond_{[13.9,44.2)}(y > 17.66)).\end{aligned} \tag{6.15}$$

In plain English, (6.15) means "At some point, between 10s and 69s in the future the output dips below 24.9 m/s and the output exceeds 17.66 m/s. between 10 and 44.2 s in the future." The formula was inferred using 15 simulated annealing cycles with 15 sample points per cycle. The computation time was 154 s on an 8 core PC with 2.1 GHz processors and 8 GB RAM.

Figure 6·14, shows the values of the scale parameters overlaid with braked and un-braked velocity outputs. Formula (6.15) is consistent with the oscillatory nature of the velocity output under normal behavior, as the velocity must increase and decrease over a window. Note again that this oscillatory behavior was recovered without any

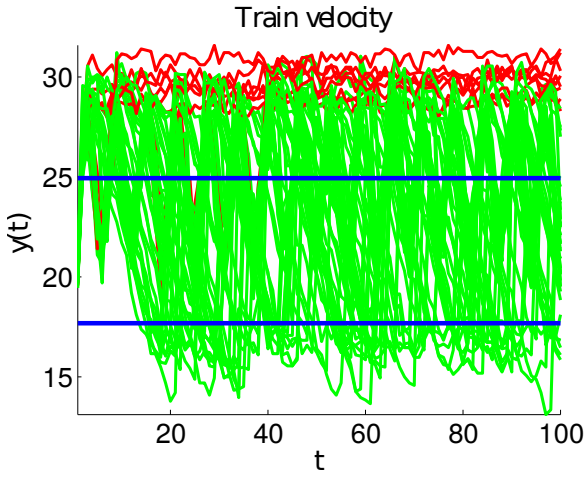


Figure 6·14: Outputs from 6·13. The green trajectories are from the system with brakes engaged and red trajectories are with brakes disengaged. The scale parameters from (6.15) are indicated with blue lines.

prior knowledge of which trajectories were good or bad, when possible attacks could happen, or what those attacks could possibly look like. The formula (6.15) perfectly separates the data, i.e. the misclassification rate is 0.

Chapter 7

Reinforcement learning and STL

In this chapter, we consider using reinforcement learning to control a system to satisfy a given temporal logic specification. In contrast to previous works on this topic, we consider STL specifications instead of LTL specifications. Thus, our approach is more appropriate for systems with real-valued state space. An example of this problem is automatically controlling an HVAC system with requirements such as "Maintain the temperature in room 1 between 68 and 74 degrees for 6 hours. If the temperature in room 2 drops below 66 degrees, ensure it is raised to 68 degrees within 10 minutes." when we do not have an accurate model of how applying heat or cooling will affect the temperature in every location in the building. We provide provably convergent algorithms based on Q-learning (Tsitsiklis, 1994) to maximize the probability of satisfaction or maximize the expected robustness with respect to a given STL formula. Further, we demonstrate that optimizing the expected robustness instead of the probability of satisfaction in some cases results in better performance of

Here, we use discrete-time signals and a discrete-time version of STL. The recursive semantics and robustness degree are calculated in the same way, except continuous intervals are now defined as finite sequences of integers.

In Section 7.1, we define a finite-memory Markov decision process abstraction we use in this work. Next, in Section 7.2, we define the problems of using reinforcement

learning to learn to control a system modeled as a τ-MDP to maximize the probability of satisfying a given STL specification or to maximize the expected robustness with respect to a given STL formula. We present our provably-convergent Q-learning algorithms in Section 7.3. Finally, we conclude with a pair of robot navigation case studies in Section 7.4.

7.1 τ-MDP

In this chapter, the desired mission specification is described by an STL fragment with the following *syntax* :

$$\begin{array}{ll} \phi & := \ \diamondsuit_{[0,T]}\psi | \ \square_{[0,T]}\psi, \\ \psi & := f(s) \leq d | \neg\xi | \xi_1 \wedge \xi_2 | \xi_1 U_{[a,b)} \xi_2, \end{array} \tag{7.1}$$

where ϕ, ψ, and ξ are all STL formulae. We denote the horizon length (Chapter 5) of the inner specification ψ as $hrz(\psi) = \tau$. Thus, in order to update at time t whether or not a formula ϕ with syntax (7.1) has been satisfied or violated, we can use the τ previous state values $s^{t-\tau+1:t}$

Example 7.1. Consider the robot navigation problem illustrated in Figure 7·1(a). The specification is "Visit Regions A or B and visit Regions C or D every 4 time units along a mission horizon of 100 units." Let $s^t = \begin{bmatrix} x^t & y^t \end{bmatrix}^T$, where x and y are the $x-$ and $y-$ components of the signal s. This task can be formulated in STL as

$$\begin{array}{lll} \phi = & \square_{[0,100)} & \psi \\ \psi = & \Big(\diamondsuit_{[0,4)} & \big((x_s > 2 \wedge x_s < 3 \wedge y_s > 2 \wedge y_s < 3) \\ & & \vee (x_s > 4 \wedge x_s < 5 \wedge y_s > 4 \wedge y_s < 5)\big) \\ & \wedge \diamondsuit_{[0,4)} & \big((x_s > 2 \wedge x_s < 3 \wedge y_s > 4 \wedge y_s < 5) \\ & & \vee (x_s > 4 \wedge x_s < 5 \wedge y_s > 2 \wedge y_s < 3)\big)\Big). \end{array} \tag{7.2}$$

□

In order to make as few assumptions as possible about the "black box" systems considered in this paper, we discretize the control space and the state space of the system. In particular, we define a finite set of control actions Act that the system can take. Then, we partition the state space of the system to form the quotient graph $\mathcal{G} = (\Sigma, E)$, where Σ is a set of discrete states corresponding to the regions of the state space and E corresponds to the set of edges such that an edge between two states σ_i and σ_j exists in E if and only if σ_i and σ_j are neighbors (share a boundary) in the partition.

Example 7.1 (Cont'd.)**.** Let the robot shown in Figure 7·1(a) evolve according to the discrete-time Dubins dynamics

$$\begin{aligned} x^{t+1} &= x^t + v\delta^t \cos\theta^t \\ y^{t+1} &= y^t + v\delta^t \sin\theta^t, \end{aligned} \tag{7.3}$$

where x^t and y^t are the x and y coordinates of the robot at time t, v is its forward speed, δ^t is a time interval, and the robot's orientation is given by θ^t. The control primitives in this case are given by $Act = \{up, down, left, right\}$ which correspond to the directions on the grid. Each (noisy) control primitive induces a distribution with support $\theta_{des} \pm \Delta\theta$, where θ_{des} is the orientation where the robot is facing the desired cell. When a motion primitive is enacted, the robot rotates to an angle θ^t drawn from the distribution and moves along that direction for δ^t time units. The partition of the state space and the induced quotient $\mathcal{G}$ are shown in Figures 7·1(a) and 7·1(b), respectively. A state $\sigma_{i,j}$ in the quotient (Figure 7·1(b)) represents the region in the partition of the state space (Figure 7·1(a)) with the point (i, j) in the lower left hand corner. □

For the case in which the stochastic dynamics of the system are known and we wish to find a policy to maximize the probability of satisfying a given LTL formula, the typical approach is to discretize the system as above and construct a Markov decision process abstraction. One approach to the problem of enforcing LTL satisfaction in a stochastic system is to partition the state state space and design control primitives that can (nominally) drive the system from one region to another region. These controllers, the stochastic dynamical model of the system, and the quotient obtained from the partition are used to construct an MDP, called a bounded parameter MDP or BMDP, whose transition probabilities are interval-valued (Abate et al., 2011). These BMDPs can then be composed with a DRA constructed from a given LTL formula to form a product interval-valued MDP. Dynamic programming (DP) can then be applied over this MDP to generate a policy from region to control primitives that maximize the probability of satisfaction. Other approaches to this problem include aggregating the states of a given quotient until an MDP can be constructed such that the transition probability can be considered constant (with bounded error) (Lahijanian et al., 2012b). The optimal policy can be computed over the resulting MDP using DP (Lahijanian et al., 2012a) or approximate DP, e.g. actor-critic methods (Ding et al., 2012). In our case, since STL has time-bounded semantics, we cannot use an automaton with a time-abstract acceptance condition (e.g. a DRA) to represent it and then compose it with a BMDP or MDP to check formula satisfaction.

For this reason, we choose to learn policies over a finite-memory MDP $\mathcal{M}_\tau$, called a τ-MDP, whose states correspond to paths of length τ in $\mathcal{G}$.

Definition 7.1. Given a quotient of a system $\mathcal{G} = (\Sigma, E)$ and a set of actions Act, a *τ-Markov Decision Process (τ-MDP)* is a tuple $\mathcal{M}_\tau = (\mathcal{S}, \sigma_\tau^0, Act, \mathcal{P})$, where

- $\mathcal{S} \subseteq (\Sigma \cup \epsilon)^\tau$ is the set of finite states, where ϵ is the empty string. Each state

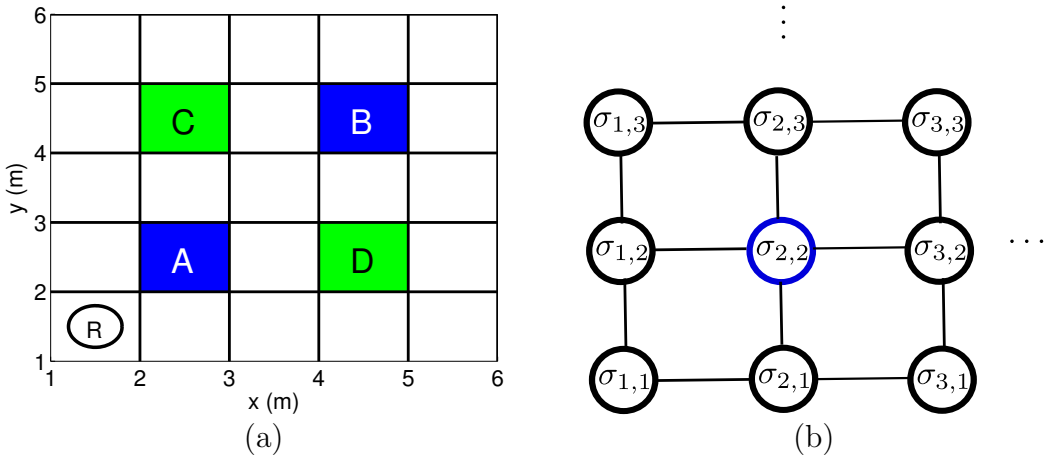


Figure 7·1: (a) Robot navigation problem with a partitioned state space. (b) Subsection of the induced quotient.

$\sigma_\tau \in \mathcal{S}$ corresponds to a $\tau-$horizon (or shorter) path in $\mathcal{G}$. Shorter paths of length $n < \tau$ (representing the case in which the system has not yet evolved for τ time steps) have ϵ prepended $\tau - n$ times.

- $\sigma^0_\tau = \epsilon^{\tau-1}\sigma^0$, where σ^0 corresponds to the region that contains the initial state of the system s^0
- $\mathcal{P} : \mathcal{S} \times Act \times \mathcal{S} \to [0, 1]$ is a probabilistic transition relation. $\mathcal{P}(\sigma_\tau, a, \sigma'_\tau)$ can be positive only if the first $\tau - 1$ states of σ'_τ are equal to the last $\tau - 1$ states of σ_τ and there exists an edge in $\mathcal{G}$ between the final state of σ_τ and the final state of σ'_τ.

□

We denote the state of the τ-MDP at time t as $\sigma^t_\tau = \sigma^{t-\tau+1:t}$. Given a trajectory $s^{t-\tau+1:t}$ of the original system, we define its induced *trace* in the τ-MDP $\mathcal{M}_\tau$ as $Tr(s^{t-\tau+1:t}) = \sigma^{t-\tau+1:t} = \sigma^t_\tau$. The construction of a τ-MDP from a given quotient and set of actions is straightforward. We make the following key assumptions on the quotient and the resulting τ-MDP:

- The partition that induces $\mathcal{G}$ is such that the defined control actions Act will drive the system either to a point in the starting region or to a point in a neighboring region of the partition, e.g. no regions are "skipped".

- The transition relation $\mathcal{P}$ is Markovian. This assumption is reasonable, as each state in the τ-MDP represents paths of length τ in $\mathcal{G}$. Thus, $\mathcal{P}_\tau$ is conditioned on the previous τ regions the system was in rather than simply the previous region.

Example 7.1 (cont'd)**.** Figure 7·2 shows a portion of the τ-MDP constructed from Figure 7·1. The states in $\mathcal{M}_4$ are labeled with the corresponding sample paths of length 4 in $\mathcal{G}$. The transitions are annotated with some example transition probabilities. The green and blue σ's in the states in $\mathcal{M}_4$ correspond to green and blue regions from Figure 7·1. □

7.2 Problem Definition

In this chapter, we address the following two problems.

Problem 7.1 (Maximum Probability of Satisfaction)**.** Let $\mathcal{M}_\tau$ be a τ-MDP abstracted from a system with unknown dynamics $s^{t+1} = f(s^t, u^t, w^t)$, where u^t is a control action and w^t is a random process. Given an STL formula ϕ with syntax (7.1), find a policy $\mu^*_{mp} \in \mathcal{F}(\mathcal{S} \times \mathbb{N}, Act)$ by solving

$$\begin{aligned} \mu^*_{mp} = & \underset{\mu \in \mathcal{F}(\mathcal{S}\times\mathbb{N}, Act)}{\arg\max} \; Pr_{s^{0:T}}[s^{0:T} \models \phi] \\ & \text{subject to} \\ & s^{t+1} = f(s^t, \mu(s^{t-\tau+1:t}, T-t), w^t) \end{aligned} \tag{7.4}$$

□

Problem 7.2 (Maximum Average Robustness)**.** Let $\mathcal{M}_\tau$ and f be as defined in Problem 7.1. Given an STL formula ϕ with syntax (7.1), find a policy $\mu^*_{mr} \in \mathcal{F}(\mathcal{S}, Act)$ by solving

$$\begin{aligned} \mu^*_{mr} = & \underset{\mu \in \mathcal{F}(\mathcal{S}\times\mathbb{N}, Act)}{\arg\max} \; E_{s^{0:T}}[r(s^{0:T}, \phi)] \\ & \text{subject to} \\ s^{t+1} = & f(s^t, \mu(s^{t-\tau+1:t}, T-t), w^t) \end{aligned} \tag{7.5}$$

□

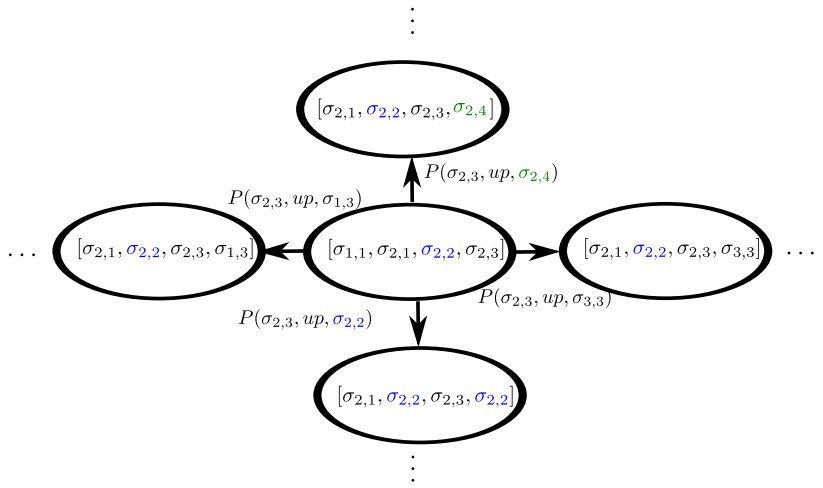


Figure 7·2: Part of the τ-MDP constructed from the robot navigation scenario shown in Figure 7·1.

Problems 7.1 and 7.2 are two alternate solutions to enforce a given STL specification. The policy found by Problem 7.1, i.e. μ^*_{mp}, maximizes the chance that ϕ will be satisfied, while the policy found by Problem 7.2, i.e. μ^*_{mr}, drives the system to satisfy ϕ as strongly as possible on average. Problems similar to (7.4) have already been considered in the literature (e.g., (Fu and Topcu, 2014; Sadigh et al., 2014)). However, Problem 7.2 is a novel formulation that provides some advantages over Problem 7.1. As we show in Section 7.2.1, for some special systems, μ^*_{mr} achieves the same probability of satisfaction as μ^*_{mp}. Furthermore, if ϕ is not satisfiable, any arbitrary policy could be a solution to Problem 7.1, as all policies will result in a satisfaction

probability of 0. If ϕ is unsatisfiable, Problem 7.2 yields a solution that attempts to get as close as possible to satisfying the formula, as the optimal solution will have an average robustness value that is least negative. This could be quite useful for practical situations when it is not know *a priori* whether a given specification can be satisfied.

The forms of the objective functions differ for the two different types of formula, $\phi = \ \diamondsuit_{[0,T)}\psi$ and $\phi = \ \square_{[0,T)}\psi$.

Case 1: Consider an STL formula $\phi = \ \diamondsuit_{[0,T)}\psi$. In this case, the objective function in (7.4) can be rewritten as

$$Pr_{s^{0:T}}[\exists t = \tau, \ldots, T-\tau \text{ s.t. } s^{t-\tau+1:t} \models \psi], \tag{7.6}$$

and the objective function in (7.5) can be rewritten as

$$E_{s^{0:T}}[\max_{t=\tau,\ldots,T-\tau} r(s^{t-\tau+1:t}, \psi)]. \tag{7.7}$$

Problem 7.1 becomes a "quasi-reachability" problem, in which the goal is to reach a τ-state σ_τ^t such that the associated trajectories satisfy ϕ with high probability. Similarly, Problem 7.2 becomes a search for a trajectory $s^{t-\tau+1:t}$ that maximizes $r(s^{t-\tau+1:t}, \phi)$.

Case 2: Now, consider an STL formula $\phi = \ \square_{[0,T)}\psi$. The objective function in (7.4) can be rewritten as

$$Pr_{s^{0:T}}[\forall t = \tau, \ldots, T-\tau, \ s^{t-\tau:t} \models \psi], \tag{7.8}$$

and the objective function in (7.5) can be rewritten as

$$E_{s^{0:T}}[\min_{t=\tau,\ldots,T-\tau} r(s^{t-\tau+1:t}, \psi)]. \tag{7.9}$$

Problem 7.1 is a "quasi-avoidance" problem, in which the goal is to avoid some $\tau-$states in the τ-MDP, whose associated trajectories violate ψ with high probability. Similarly, Problem 7.2 is the problem of minimizing the "worst-case" violation of ψ over the horizon T.

7.2.1 Relationship Between Maximizing Expected Robustness and Maximizing Probability of Satisfaction

Here, we demonstrate that the solution to (7.5) subsumes the solution to (7.4) for a certain class of systems. For the rest of this section, we only consider formulae of the type $\phi = \ \diamondsuit_{[0,t)}\psi$. Let $\mathcal{M}_\tau = (\mathcal{S}, \mathcal{P}_\tau, Act)$ be a τ-MDP. For simplicity, we make the following assumption on $\mathcal{S}$.

Assumption 7.1. *For every state $\sigma_\tau \in \mathcal{S}$, either every trajectory $s^{t+\tau-1:t}$ that passes through the sequence of regions $\sigma^{t+\tau-1:t}$ associated with σ_τ satisfies ψ, denoted $\sigma_\tau \models \psi$, or every trajectory that passes through the sequence of regions associated with σ_τ does not satisfy ψ, denoted $\sigma_\tau \not\models \psi$.* □

Assumption 7.1 can be enforced in practice during partitioning. Further, we define the set A as $A = \{\sigma_\tau \in \mathcal{S} | \sigma_\tau \models \psi\}$.

Definition 7.2. The signed distance of a $\tau-$state $\sigma_\tau^i \in \mathcal{S}$ to a set $X \subseteq \mathcal{S}$ is

$$d(\sigma_\tau^i, X) = \begin{cases} \min\limits_{\sigma_\tau^j \in X} l(\sigma_\tau^i, \sigma_\tau^j) & \sigma_\tau^i \notin X \\ -\min\limits_{\sigma_\tau^j \in \mathcal{S}\setminus X} l(\sigma_\tau^j, \sigma_\tau^i) & \sigma_\tau^i \in X \end{cases} \tag{7.10}$$

where $l(\sigma_\tau^i, \sigma_\tau^j)$ is the length of the shortest path from σ_τ^i to σ_τ^j. □

Assumption 7.2. *Let $D_{\sigma_\tau}(\delta) = Pr[r(s^{t:t+\tau}, \psi) > \delta | Tr(s^{t:t+\tau}) = \sigma_\tau]$. For any two states,*

$$d(\sigma_\tau^i, A) < d(\sigma_\tau^j, A) \Rightarrow D_{\sigma_\tau^i}(\delta) \geq D_{\sigma_\tau^j}(\delta) \ \forall \delta \in [R_{min}, R_{max}] \tag{7.11}$$

□

Finally, we make the following simple assumption.

Assumption 7.3. *For any signal $s^{t-\tau+1:t}$ such that $Tr(s^{t-\tau+1:t}) \in \mathcal{S}$, let $r(s^{t-\tau+1:t}, \psi)$ be bounded from below by R_{min} and from above by R_{max}.* □

Now we define the policies μ^*_{mp} and μ^*_{mr} over $\mathcal{M}_\tau$ as

$$\begin{aligned} \mu^*_{mp} = & \underset{\mu \in \mathcal{F}(\mathcal{S}\times\mathbb{N}, Act)}{\arg\max} \; Pr_{\sigma_\tau^{0:T}}\Big[\exists t \in [0,T] \text{ s.t. } \sigma_\tau^t \models \psi\Big] \\ & \text{subject to} \\ & \sigma_\tau^{t+1} = \sigma_\tau \text{ w. p. } \mathcal{P}(\sigma_\tau^t, \mu(\sigma_\tau^t, T-t), \sigma_\tau), \end{aligned} \tag{7.12}$$

$$\begin{aligned} \mu^*_{mr} = & \underset{\mu \in \mathcal{F}(\mathcal{S}\times\mathbb{N}, Act)}{\arg\max} \; E_{\sigma_\tau^{0:T}}\Big[\max_{t=0,\ldots,T} r(\sigma_\tau^t, \psi)\Big] \\ & \text{subject to} \\ & \sigma_\tau^{t+1} = \sigma_\tau \text{ w. p. } \mathcal{P}(\sigma_\tau^t, \mu(\sigma_\tau^t, T-t), \sigma_\tau). \end{aligned} \tag{7.13}$$

Proposition 7.1. *Let Assumptions 7.1,7.2 and 7.3 hold. The policy μ^*_{mr} maximizes the expected probability of satisfaction.*

Proof. Given any policy μ, its associated reachability probability can be defined as

$$Pr_\mu(\sigma_\tau) = Pr_\mu[\sigma_\tau = \underset{\sigma_\tau^0,\ldots,\sigma_\tau^{T-\tau}}{\arg\min}\, d(\sigma_\tau, A)]. \tag{7.14}$$

By definition, the expected probability of satisfaction for a given policy μ is

$$\begin{aligned} EPS(\mu) &= E\big[I(\exists 0 < k < T-\tau \text{ s.t. } \sigma_\tau^k \models \psi)\big] \\ &= \sum_{\sigma_\tau \in \mathcal{S}} P^r_\mu(\sigma_\tau) I(\sigma_\tau \in A) \\ &= \sum_{\sigma_\tau \in A} P^r_\mu(\sigma_\tau). \end{aligned} \tag{7.15}$$

Thus, the expected robustness of policy μ becomes

$$\begin{aligned} ER(\mu) &= E\big[\max_{k=0,\ldots,T-\tau} r(\sigma_\tau^k, \psi)\big] \\ &= \int_0^{R_{max}} Pr\big[\max_{k=0,\ldots,T-\tau} r(\sigma_\tau^k, \psi) > x\big]dx \\ &\quad + \int_{R_{min}}^0 1 - Pr\big[\max_{k=0,\ldots T-\tau} r(\sigma_\tau^k, \psi) > x\big]dx \\ &= \int_{R_{min}}^{R_{max}} Pr\big[\max_{k=0,'ldots,T-\tau} r(\sigma_\tau^k, \psi) > x\big]dx + R_{min} \\ &= \int_{R_{min}}^{R_{max}} \sum_{\sigma_\tau \in \mathcal{S}} P_\mu^r(\sigma_\tau) D_{\sigma_\tau}(x)dx + R_{min} \\ &= \sum_{\sigma_\tau \in A} P_\mu^r(\sigma_\tau) \int_0^{R_{max}} D_{\sigma_\tau}(x)dx + \\ &\quad \sum_{\sigma_\tau \notin A} P_\mu^r(\sigma_\tau) \int_{R_{min}}^0 D_{\sigma_\tau}(x)dx + R_{min}. \end{aligned} \tag{7.16}$$

Since R_{min} is constant, maximizing (7.16) is equivalent to

$$\begin{gathered} \max_\mu \sum_{\sigma_\tau \in A} Pr_\mu(\sigma_\tau) \int_0^{R_{max}} D_{\sigma_\tau}(x)dx + \sum_{\sigma_\tau \notin A} Pr_\mu(\sigma_\tau) \int_{R_{min}}^0 D_{\sigma_\tau}(x)dx \\ \text{subject to} \\ \sum_{\sigma_\tau \in A} Pr_\mu(\sigma_\tau) = p \end{gathered} \tag{7.17}$$

We can rewrite the objective as

$$\begin{aligned} J(\mu) &= p \sum_{\sigma_\tau \in A} Pr_\mu\big[\sigma_\tau = \arg\min_{\sigma_\tau^0,\ldots,\sigma_\tau^{T-\tau}} d(\sigma_\tau, A) | \sigma_\tau \in A\big] \int_0^{R_{max}} D_{\sigma_\tau}(x)dx \\ &\quad + (1-p) Pr_\mu\big[\sigma_\tau = \arg\min_{\sigma_\tau^0,\ldots,\sigma_\tau^{T-\tau}} d(\sigma_\tau, A) | \sigma_\tau \notin A\big] \int_{R_{min}}^0 D_{\sigma_\tau}(x)dx \end{aligned} \tag{7.18}$$

Now,

$$\begin{aligned} \frac{\partial J(\mu)}{\partial p} &= \sum_{\sigma_\tau \in A} Pr_\mu\big[\sigma_\tau = \arg\min_{\sigma_\tau^0,\ldots,\sigma_\tau^{T-\tau}} d(\sigma_\tau, A) | \sigma_\tau \in A\big] \int_0^{R_{max}} D_{\sigma_\tau}(x)dx \\ &\quad - Pr_\mu\big[\sigma_\tau = \arg\min_{\sigma_\tau^0,\ldots,\sigma_\tau^{T-\tau}} d(\sigma_\tau, A) | \sigma_\tau \notin A\big] \int_{R_{min}}^0 D_{\sigma_\tau}(x)dx \\ &\geq 0 \end{aligned} \tag{7.19}$$

Thus, if we increase the allowed acceptance probability p, the expected robustness is non-decreasing. Therefore, the policy that maximizes the robustness also achieves

the maximum satisfaction probability. □

7.3 Policy Generation through Q-Learning

Since we do not know the dynamics of the system under control, we cannot *a priori* predict how a given control action will affect the evolution of the system and hence its progress towards satisfying/dissatisfying a given specification. Thus, we use the well-known paradigm of *reinforcement learning* to learn policies to solve Problems 7.1 and 7.2. In reinforcement learning, the system takes actions and records the rewards associated with the state-action pair. These rewards are then used to update a feedback policy that maximizes the expected gathered reward. In our cases, the rewards that we collect over $\mathcal{M}_\tau$ are related to whether or not ψ is satisfied (Problem 7.1) or how robustly ψ is satisfied/violated (Problem 7.2).

Our solutions to these problems rely on a Q-learning formulation (Tsitsiklis, 1994). Let $R(\sigma_\tau^t, a)$ be the reward collected when action $a \in Act$ was taken in state $\sigma_\tau^t \in \mathcal{S}$. Define the function $Q : \mathcal{S} \times Act \times \mathbb{N}$ as

$$\begin{aligned} Q(\sigma_\tau^{T-t}, a, t) &= R(\sigma_\tau^{T-t}, a) + \max_{\{\mu_l \in\}_{l=T-t-1}^{T}} E\big[\textstyle\sum_{l=T-t-1}^{T} R(\sigma_\tau^l, \mu_l(\sigma_\tau^l))\big] \\ &= R(\sigma_\tau^{T-t}, a) + \max_{a' \in Act} Q(\sigma_\tau^{T-t+1}, a', t-1). \end{aligned} \tag{7.20}$$

For an optimization problem with a cumulative objective function of the form

$$\sum_{l=\tau:T} R(\sigma_\tau^l, a^l), \tag{7.21}$$

the optimal policy $\mu^* \in \mathcal{F}(\mathcal{S}, Act)$ can be found by

$$\mu^*(\sigma_\tau^t, T-t) = \arg\max_{a \in Act} Q(\sigma_\tau^t, a, T-t). \tag{7.22}$$

Applying the update rule

$$\begin{aligned} Q_{t+1}(\sigma_\tau^t, a^t, T-t) = \quad & (1-\alpha_t) Q_t(\sigma_\tau^t, a^t, T-t) + \\ & \alpha_t [R(\sigma_\tau^t, a^t) + \gamma \max_{a' \in A} Q_t(\sigma_\tau^{t+1}, a')] \end{aligned} \tag{7.23}$$

will cause Q_t converges to Q w.p. 1 as t goes to infinity (Tsitsiklis, 1994).

7.3.1 Batch Q-learning

We cannot reformulate Problems 7.1 and 7.2 into the form (7.21) (see Section 7.2). Thus, we propose an alternate $Q-$learning formulation, called batch Q-learning, to solve these problems. Instead of updating the Q-function after each action is taken, we wait until an entire episode $s^{[0:T)}$ is completed before updating the Q-function. The batch Q-learning procedure is summarized in Algorithm 7.1.

Algorithm 7.1 The Batch Q learning algorithm.

function BatchQLearn(Sys,probType,N_{ep},ϕ)
$Q \leftarrow$ RandomInitialization
$\mu \leftarrow$ InitializePolicy(Q)
for $n = 1$ to N_{ep} **do**
  $s^{[0,T)} \leftarrow$Simulate(Sys, μ)
  $Q \leftarrow$ UpdateQFunction(Q,μ,$s^{0:T}$,ϕ,probType)
  $\mu \leftarrow$ UpdatePolicy(μ, Q)
return Q,μ

As in the typical Q-learning approach, Q is initialized to random values and μ is computed from the initial Q values. Then, for N_{ep} episodes, the system is simulated using the current policy μ. The observed trajectory is then used to update the Q function according to Algorithm 7.2. The new value of the Q function is used to update the policy μ. Note that μ is not always the arg max of Q, as the policy update function may be randomized to allow exploration of the policy space.

Algorithm 7.2 Function used to update Q function used in Algorithm 7.1.

function UpdateQFunction(Q,μ,$s^{0:T}$,ϕ,γ,probType)
for $n = T - \tau - 1$ to τ **do**
  if probType is MaximumProbability **then**
    if $\phi = \ \diamondsuit_{[0,T)}\psi_\tau$ **then**
      $Q_{tmp}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n)) \leftarrow \max(I(s^{n-\tau+1:n} \models \phi),\ \gamma Q_{tmp}(\sigma_\tau^{n+1}, \mu(\sigma_\tau^{n+1}, T - n - 1))$
    else
      $Q_{tmp}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n)) \leftarrow \min(-I(s^{n-\tau+1:n} \models \phi),\ \gamma Q_{tmp}(\sigma_\tau^{n+1}, \mu(\sigma_\tau^{n+1}, T - n - 1))$
  else
    if $\phi = \ \diamondsuit_{[0,T)}\psi_\tau$ **then**
      $Q_{tmp}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n)) \leftarrow \max(r(s^{n-\tau+1:n}, \phi), \gamma Q_{tmp}(\sigma_\tau^{n+1}, \mu(\sigma_\tau^{n+1}, T-n-1))$
    else
      $Q_{tmp}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n)) \leftarrow \min(-r(s^{n-\tau+1:n}, \phi), \gamma Q_{tmp}(\sigma_\tau^{n+1}, \mu(\sigma_\tau^{n+1}, T - n - 1))$
  $Q_{new}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n) \leftarrow (1-\alpha) Q_{tmp}(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n) + \alpha Q(\sigma_\tau^n, \mu(\sigma_\tau^n, T-n)$
return Q_{new}

7.3.2 Convergence of Batch Q-learning

Given a formula of the form $\phi = \ \diamondsuit_{[0,T)}\psi_\tau$ and an objective of maximizing the expected robustness (Problem 7.2), we show that applying Algorithm 7.1 converges to the optimal solution. The other three cases discussed in Section 7.2 can be proven similarly. The following analysis is based on (Melo,).

Define the reward-to-go (derived from (7.7)) at time k as

$$V(\sigma_\tau^k, \{a^t\}_{t=k}^{T-1}, T-k) = E[\max_{t=k,\ldots,T-1} \gamma r(\sigma_\tau^t, \psi)], \tag{7.24}$$

where $0 < \gamma < 1$ is a discount factor. The optimal (discounted) reward to go is

$$\begin{aligned} V^*(\sigma_\tau^k, T-k) &= \max_{\{a^t\}_{t=k}^{T-1}} V(\sigma_\tau^k, \{a^t\}_{t=k}^{T-1}, T-k) \\ &= \max_{a \in Act} \textstyle\sum_{\sigma_\tau^{t+1}} P(\sigma_\tau^t, a, \sigma^{t+1}) \\ &\quad \times \max(r(\sigma_\tau^t, \psi), \gamma V^*(\sigma_\tau^{t+1}, T-k-1)). \end{aligned} \tag{7.25}$$

The associated optimal Q function is

$$Q^*(\sigma_\tau^k, a, T-k) = \begin{array}{l} \sum_{\sigma_\tau^{t+1}} P(\sigma_\tau^t, a, \sigma^{t+1}) \max(r(\sigma_\tau^t, \psi), \\ \max\limits_{b \in Act} \gamma Q^*(\sigma_\tau^{t+1}, b, T-t-1)). \end{array} \tag{7.26}$$

Proposition 7.2. *The optimal Q-function given by* (7.26) *is a fixed point of the contraction mapping H where*

$$(Hq)(\sigma_\tau^t, a, T-t) = \begin{array}{l} \sum_{\sigma_\tau^{t+1}} P(\sigma_\tau^t, a, \sigma^{t+1}) \max(r(\sigma_\tau^t, \psi), \\ \gamma \max\limits_{b \in Act} q(\sigma_\tau^{t+1}, b, T-t-1)). \end{array} \tag{7.27}$$

Proof. By (7.26), if H is a contraction mapping, then Q^* is a fixed point of H. Consider

$$\begin{array}{rl} ||Hq_1 - Hq_2||_\infty & = \max\limits_{\sigma_\tau, a} \sum\limits_{\sigma'_\tau} P(\sigma_\tau, a, \sigma'_\tau)(\max(r(\sigma_\tau, \psi), \\ & \gamma \max\limits_{b \in Act} q_1(\sigma'_\tau, b, T-t-1)) \\ & - \max(r(\sigma_\tau, \psi), \gamma \max\limits_{b \in Act} q_2(\sigma'_\tau, b, T-t-1)). \end{array} \tag{7.28}$$

Define

$$q_j^*(t) = \max_{b \in Act} \gamma q_1(\sigma'_\tau, b, t). \tag{7.29}$$

WOLOG let $q_1^*(T-t-1) > q_2^*(T-t-1)$. Define

$$R(\sigma'_\tau) = (\max(r(\sigma_\tau, \psi), q_1^*(T-t-1) - \max(r(\sigma_\tau, \psi), q_2^*(T-t-1)) \tag{7.30}$$

There exist 3 possibilities for the value of $R(\sigma_\tau)$.

$$\begin{array}{l} r(\sigma_\tau, \psi) > q_1^*(T-t-1) > q_2^*(T-t-1) \\ \Rightarrow R(\sigma'_\tau) = 0 \end{array}. \tag{7.31a}$$

$$\begin{array}{l} q_1^*(T-t-1) > r(\sigma_\tau, \psi) > q_2^*(T-t-1) \\ \Rightarrow R(\sigma'_\tau) = ||q_1^*(T-t-1) - r||_\infty < \gamma ||q_1 - q_2||_\infty \end{array} \tag{7.31b}$$

$$\begin{array}{l} q_1^*(T-t-1) > q_2^*(T-t-1) > r(\sigma_\tau, \psi) \\ \Rightarrow R(\sigma'_\tau) < \gamma ||q_1 - q_2||_\infty. \end{array} \tag{7.31c}$$

Thus, this means that $R(\sigma'_\tau) \leq \gamma ||q_1 - q_2||_\infty \ \forall \sigma'_\tau$. Hence,

$$\begin{array}{ll} ||Hq_1 - Hq_2||_\infty & = \max_{\sigma_\tau, a} \sum_{\sigma'_\tau} P(\sigma_\tau, a, \sigma'_\tau) R(\sigma'_\tau) \\ & \leq \max_{\sigma_\tau, a} \sum_{\sigma'_\tau} P(\sigma_\tau, a, \sigma'_\tau) \gamma ||q_1 - q_2||_\infty \\ & \leq \gamma ||q_1 - q_2||_\infty. \end{array} \tag{7.32}$$

Therefore, H is a contraction mapping. □

The proof of Q learning convergence relies on the following result from stochastic systems (Jaakkola et al., 1994).

Lemma 7.1. *The random process Δ^t defined as*

$$\Delta^{t+1} = (1 - \alpha^t)\Delta^t + \alpha^t F^t \tag{7.33}$$

converges to 0 w.p.1 under the assumptions

1. $0 < \alpha^t < 1, \sum_{t=0}^{\infty} \alpha^t = \infty$, *and* $\sum_{t=0}^{\infty} (\alpha^t)^{(2)} < \infty$
2. $||E[F^t|\mathcal{F}^t]||_W \leq \gamma ||\Delta^t||_W$ *where* $\gamma < 1$
3. $var[F^t|\mathcal{F}^t] \leq C(1 + ||\Delta^t||_W^{(2)})$

Proposition 7.3. *The Q-learning rule given by*

$$\begin{array}{ll} Q^{t+1}(\sigma_\tau^k, a^k, T-k) = & (1-\alpha^t) Q^t(\sigma_\tau^k, a^k, T-k) \\ & + \alpha^t \max(r(\sigma_\tau^k, \psi), \underset{b \in Act}{max} \gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)), \end{array} \tag{7.34}$$

converges to the optimal Q function (7.26) *if the sequence $\{\alpha^t\}_{t=0}^{\infty}$ is such that*

$$\sum_{t=0}^{\infty} \alpha^t = \infty \tag{7.35}$$

and

$$\sum_{t=0}^{\infty}(\alpha^t)^{(2)} < \infty \tag{7.36}$$

Proof. Define the stochastic process Δ^t as

$$\Delta^t(\sigma_\tau^k, a, T-k) = Q^t(\sigma_\tau^k, a, T-k) - Q^*(\sigma_\tau^k, a, T-k). \tag{7.37}$$

If we subtract $Q^*(\sigma_\tau^k, a, T-k)$ from both sides of (7.34), we can rewrite it as

$$\begin{aligned}\Delta^{t+1}(\sigma_\tau^k, a^k, T-k) = \quad & (1-\alpha^t)\Delta^t(\sigma_\tau^k, a^k, T-k) + \\ & \alpha^t(\max(r(\sigma_\tau^k, \psi), \max_{b\in Act}\gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)) \\ & -Q^*(\sigma_\tau^k, a, T-k))\end{aligned} \tag{7.38}$$

Now, define the stochastic process F^t as

$$\begin{aligned}F^t(\sigma_\tau^k, a, T-k-1) = \quad & \max(r(\sigma_\tau^k, \psi), \max_{b\in Act}\gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)) \\ & -Q^*(\sigma_\tau^k, a, T-k)\end{aligned} \tag{7.39}$$

such that

$$\Delta^{t+1}(\sigma_\tau^k, a^k, T-k) = \quad (1-\alpha^t)\Delta^t(\sigma_\tau^k, a^k, T-k) + \alpha^t F^t(\sigma_\tau^k, a, T-k-1) \tag{7.40}$$

Now,

$$\begin{aligned}E[F^t(\sigma_\tau^k, a^k, T-k)|\mathcal{F}^t] \quad & = \textstyle\sum_{\sigma'_\tau} P(\sigma_\tau^k, a^k, \sigma'_\tau)\max(r(\sigma_\tau^k, \psi), \\ & \max_{b\in Act}\gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)) \\ & -Q^*(\sigma_\tau^k, a, T-k)) \\ & = HQ^t(\sigma_\tau^k, a^k, T-k) - Q^*(st^k, a^k, T-k)\end{aligned} \tag{7.41}$$

From Proposition 7.2, $Q^* = HQ^*$. Hence,

$$\begin{aligned}E[F^t(\sigma_\tau^k, a^k, T-k)|\mathcal{F}^t] \quad & = HQ^t(\sigma_\tau^k, a^k, T-k) - HQ^*(st^k, a^k, T-k) \\ & \leq \gamma||Q^t - Q^*||_\infty = \gamma||\Delta^t||_\infty\end{aligned} \tag{7.42}$$

Now,

$$\begin{aligned}
&var[F^t(\sigma_\tau^k, a^k, T-k)|\mathcal{F}^t] \\
&= E[(\max(r(\sigma_\tau^k, \psi), \max_{b \in Act} \gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)) - Q^*(\sigma_\tau^k, a, T-k) \\
&- HQ^t(\sigma_\tau^k, a^k, T-k) + Q^*(st^k, a^k, T-k))^2] \\
&= E[(\max(r(\sigma_\tau^k, \psi), \max_{b \in Act} \gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1)) - HQ^t(\sigma_\tau^k, a^k, T-k))^{(2)}] \\
&= var[\max(r(\sigma_\tau^k, \psi), \max_{b \in Act} \gamma Q^t(\sigma_\tau^{k+1}, b, T-k-1))|\mathcal{F}^t].
\end{aligned} \tag{7.43}$$

Since the robustness degree with respect to a sub-trajectory of length τ is bounded, $\max_{\sigma_\tau \in \mathcal{S}} r(\sigma_\tau, \psi)$ is also bounded and hence for some value C,

$$var[F^t(\sigma_\tau^k, a^k, T-k)|\mathcal{F}^t] \leq C(1 + ||\Delta^t||_\infty^{(2)}). \tag{7.44}$$

Thus, the defined values of Δ^t and F^t and $\{\alpha^t\}$ (by assumption) adhere to the assumptions of Lemma 7.1, which means

$$\begin{aligned}
&\Delta^t(\sigma_\tau^k, a^k, T-k) \to 0 \\
&\Rightarrow Q^t(\sigma_\tau^k, a, T-k) - Q^*(\sigma_\tau^k, a, T-k) \to 0 \\
&\Rightarrow Q^t(\sigma_\tau^k, a, T-k) \to Q^*(\sigma_\tau^k, a, T-k)
\end{aligned} \tag{7.45}$$

with probability 1.

□

7.4 Case Studies

We implemented the batch-Q learning algorithm (Algorithm 7.1) and applied it to two case studies that adapt the robot navigation model from Example 7.1. For each case study, we solved Problems 7.1 and 7.2 and compared the performance of the resulting policies. All simulations were implemented in Matlab and performed on a PC with a 2.6 GHz processor and 7.8 GB RAM.

7.4.1 Case Study 1: Reachability

First, we consider a simple reachability problem. The given STL specification is

$$\phi_{cs1} = \ \diamondsuit_{[0,20)}(\ \square_{[0,4)}\xi_A \vee \ \square_{[0,4)}\xi_B), \tag{7.46}$$

where ξ_A and ξ_B are the subformulae from (7.2) which specify being in regions A and B, respectively. In plain English, (7.46) can be stated as "Within 20 time units, reach either region A or B and remain there for at least four time units." The results from applying Algorithm 7.1 are summarized in Figure 7·3, where regions A and B are in blue. We used the parameters $\gamma = 1, \alpha_t = 0.95$, $N_{ep} = 1000$ and $\epsilon^t = 0.995^t$, where ϵ^t is the probability at iteration t of selecting an action at random. Constructing the τ-MDP took 16.7s. Algorithm 7.1 took 461s to solve Problem 7.1 and 442s to solve Problem 7.2.

The two approaches perform very similarly. In the first row, we show a histogram of the robustness of 500 trials generated from the system simulated using each of the trained policies after learning has completed, i.e. without the randomization that is used during the learning phase. Note that both trained policies satisfied the specification with probability 1. The performance of robustness maximization is negligibly better, as the mean robustness is 0.0717 with standard deviation 0.0661, while the probability maximization has mean robustness 0.0716 with standard deviation 0.0617. In the second row, we see trajectories simulated by each of the trained policies. Each of the trajectories satisfies the specification and appears to be similar.

The similarity of the solutions in this case study are not surprising. If the state of the system is deep within A or B, then the probability that it will remain inside that region in the next 3 time steps (satisfy ϕ) is higher than if it is at the edge of the region. Trajectories that remain deeper in the interior of region A or B also

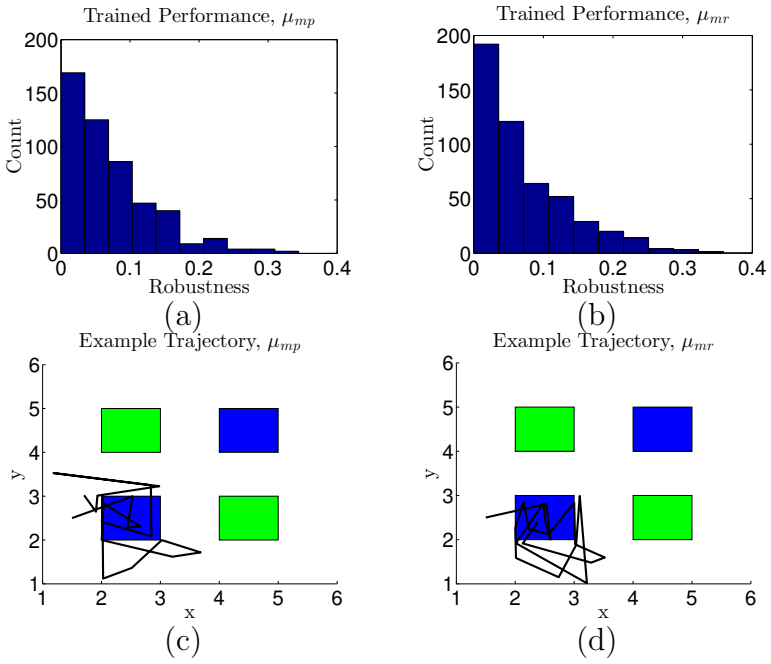


Figure 7·3: Comparison of Policies for Case Study 1. Regions A and B are in blue. Histogram of robustness values for trained policies for solution to (a) Problem 7.1 and (b) Problem 7.2. Trace generated from policies for solution to (c) Problem 7.1 and (d) Problem 7.2.

have a high robustness value. Thus, for this particular problem, there is an inherent coupling between the policies that satisfy the formula with high probability and those that satisfy the formula as robustly as possible on average.

7.4.2 Case Study 2: Repeated Satisfaction

In this second case study, we look at a problem involving repeatedly satisfying a condition finitely many times. The specification of interest is

$$\phi_{cs2} = \Box_{[0,20)}(\diamondsuit_{[0,5)}(\xi_A \vee \xi_B) \wedge \diamondsuit_{[0,5)}(\xi_C \vee \xi_D)), \tag{7.47}$$

where ξ_A through ξ_D are the subformulae from (7.2) which specify being in regions A through D. In plain English, (7.47) is "Ensure that every 5 time units over a 20 unit interval, at least one of regions A or B is entered and at least one of regions C or D is entered." Results from this case study are shown in Figure 7·4. A and B are in blue, and C and D in green. We used the same parameters from Section 7.4.1, except N_{ep}=500. Constructing the τ-MDP took 103s. Applying Algorithm 7.1 took 820s for Problem 7.1 and 844s for Problem 7.2.

In the first row, we see that the solution to Problem 7.1 satisfies the formula with probability 0 while the solution to Problem 7.2 satisfies the formula with probability 1. At first, this seems counterintuitive, as Proposition 7.3 indicates that a policy that maximizes probability would achieve a probability of satisfaction at least as high as the policy that maximizes the expected robustness. However, this is only guaranteed with an infinite number of learning trials, while here we train with relatively few trials. The performance in terms of robustness is obviously better for the robustness maximization (mean 0.0730, standard deviation 0.0699) than for the probability maximization (mean -0.3035, standard deviation 0.1962). In the second row, we see

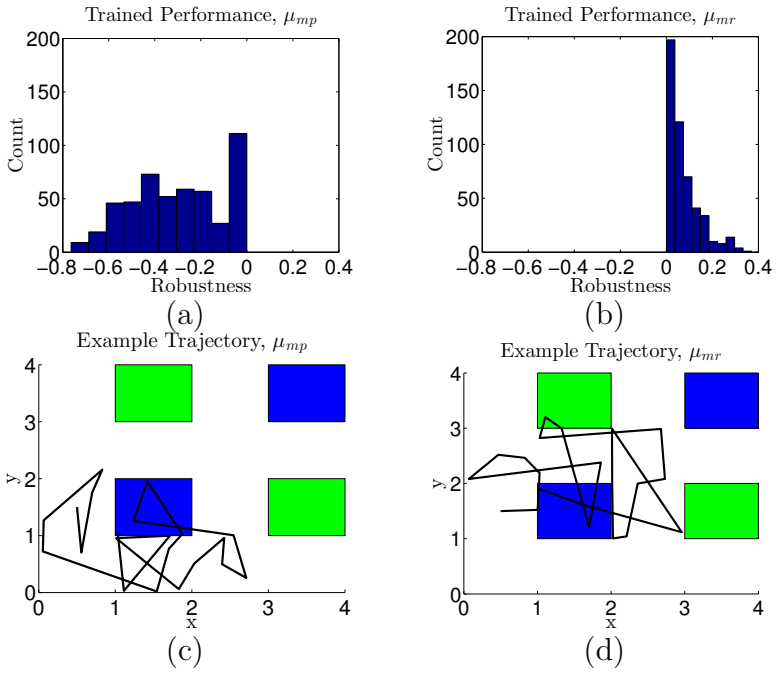


Figure 7·4: Comparison of Policies for Case Study 2. Regions A and B are shown in blue, and C and D in green. The subplots have the same meaning as in Figure 7·3.

that the trajectory produced by the solution to Problem 7.1 wanders around region A, while the trajectory generated from the solution to Problem 7.2 clearly exhibits the behavior of oscillating between blue and green regions implied by (7.47).

The discrepancy between the two solutions can be explained by what happens when trajectories that almost satisfy (7.47) occur during training. If a trajectory that almost oscillates between a blue and green region every five seconds is encountered when solving Problem 7.1, it collects 0 reward. The policy that produced this trajectory is reinforced as much as a policy that produces a trajectory that never enters any region A through D. On the other hand, when solving Problem 7.2, the policy that produces the almost oscillatory trajectory will be reinforced much more strongly, as the resulting robustness is less negative. This illustrates an important advantage of using the robustness degree as an objective in reinforcement learning. Since the robustness degree gives "partial credit" for trajectories that are close to

satisfying the policy, the reinforcement learning algorithm performs a directed search to find policies that satisfy the formula. Since probability maximization gives no partial credit, the reinforcement learning algorithm is essentially performing a random search until it encounters a trajectory that satisfies the given formula. Therefore, if the family of policies that satisfy the formula with high probability is small, it will on average take the Q-learning algorithm solving Problem 7.1 a very long time to converge to a solution that satisfies the formula with positive probability.

Chapter 8

Conclusions and Future Research Directions

In this dissertation, we presented a body of work that addresses problems involving describing, discovering, and enforcing high-level system behaviors when limited *a priori* information is available. The procedures we develop combine features from formal methods, the field concerned with verifying, enforcing, and inferring high-level specifications, with principles from estimation and machine learning, two fields which are concerned with using noisy data to discover *a priori* unknown information. The key features of our methods are their ability to handle rich, temporally layered and history dependent system properties and their ability to adapt to new information over time. Our work contributes to formal methods by reducing the amount of prior information that is required in order to be executed effectively. Our work contributes to the field of active estimation by augmenting the class of constraints that can be handled. Finally, our work contributes the field of machine learning by introducing a rich set of data classifiers and a rich set of behaviors that can be generated with an unknown system model.

This dissertation addressed a total of four different problems. The first problem was controlling a robot to gather as much information about its environment as possible while still satisfying constraints on its motion given as propositional temporal

logic formulae. An example of this problem is using an aerial robot to monitor the location of forest fires while ensuring that it periodically visits recharging stations and data upload stations and perpetually avoids restricted areas. In Chapter 3, we presented algorithms to solve this problem with finite-horizon constraints (Jones et al., 2013b; Jones et al., 2015b) and infinite-horizon constraints (Jones et al., 2015c). We developed receding horizon algorithms that are computationally efficient, are guaranteed to satisfy the given specification, and locally minimize uncertainty. For the infinite-horizon case, we introduced a high-level planner that attempts to mitigate the myopia inherent in receding horizon formulations. Our algorithms were evaluated in simulation and the finite-horizon solution was evaluated using an aerial agent with a downward-facing camera in a laboratory environment.

This research leads to several intriguing extensions. Recently, we have extended the finite-horizon problem to a multi-agent setting (Leahy et al., 2015). This work combines the receding horizon information gathering algorithm with recently- developed task distribution techniques (Chen et al., 2013a). We plan to extend this to the multi-agent, infinite horizon case. We have also recently developed a new logic for the coordination of teams of agents that we call Braid temporal logic (Diaz-Mercado et al., 2015). This logic can be used to specify properties involving the agents' relative locations, absolute location, and interactions (communications) between agents. This logic is intended to coordinate teams of vehicles to solve coordinated sensing tasks. We plan to incorporate information gathering algorithms with this new paradigm.

The second problem we addressed was developing a new specification language for partially observable systems. This new logic, called distribution temporal logic, was presented in Chapter 4 and is defined over sequences in the belief space of a discrete random variable (Jones et al., 2013a). We defined a particular instantiation of distribution temporal logic for finite-time executions. This logic is more expressive

than the previously proposed specification language developed to express properties over partially observable models. We developed a verification algorithm that calculates *ex post facto* with what probability a given execution of a partially observable system satisfies a given specification. We then used this in a statistical procedure to characterize and compare the performance of two different control strategies.

In the future, we plan to synthesize control policies that maximize the probability of satisfying a given specification. One avenue of research that we are pursuing is to construct a distribution temporal logic based on Gaussian distributions. This logic has the advantage of being able to be interpreted over systems with a continuous hidden state space. Further, since the Gaussian is a parameterized and well-studied distribution, computing functions of Gaussians is efficient even for large state spaces. We plan to combine this logic with belief-space sampling techniques, such as the recently-developed feedback information roadmap (Agha-mohammadi et al., 2011), to develop efficiently computable control policies that can steer a robot to satisfy a given distribution temporal logic specification. A simple example of this problem is steering a robot to move between several regions in an environment while continually maintaining its localization uncertainty below a pre-specified level.

The third problem we addressed in Chapter 6 was learning specifications from system execution data. We mapped a pair of problems in cyber-physical system analysis to machine-learning like problems where we learned data classifiers in the form of temporal logic formulae. We considered the problem of learning a formula that separates desirable from undesirable behavior via supervised learning (Kong et al., 2014) and the problem of learning a formula that separates anomalous data from mainstream data via unsupervised learning (Jones et al., 2014). In order to do this, we defined a fragment of temporal logic formula structures that could be organized into a directed acyclic graph based on their inclusivity. This allowed us to develop

algorithms that used a discrete search over the graph to find formula structures and use continuous optimization methods to fit parameters to these structures. We also developed an on-line version of the supervised learning algorithm based on stochastic gradient descent (Kong et al., 2015). We evaluated our methods on case studies using academic and data-driven simulations.

In the future, we plan to grow the class of formulae that can be inferred by our method. Thus far, we have only considered formulae involving rectangular predicates of the form $x < \pi$ where x is a variable and π is a constant. We would like to extend our formula structure search to consider searching over a larger sets of predicates, e.g. linear or polynomial predicates. We may find inspiration in kernelization techniques from machine learning during this process. We would also like to test our methods on data from larger systems such as electrical grids. In addition, we plan to address the problem of on-line unsupervised learning. This is considerably more difficult than on-line supervised monitoring.

The final problem we addressed was controlling a system with unknown dynamics to satisfy a given temporal logic specification. We extended this problem from finite systems and propositional temporal logic constraints to continuous systems with predicate temporal logic constraints. We defined two different ways to enforce the given specification: maximizing the probability of satisfying the specification and maximizing the expected robustness, a continuous measure of how strongly a given trajectory satisfies or violates the specification. We showed that in some cases, maximizing the expected robustness subsumes maximizing the probability of satisfaction. We developed novel, provably convergent Q-algorithms to solve both problems. In simulation, we showed that maximizing the expected robustness performed better in terms of both probability and robustness than maximizing the probability with a low number of training samples.

In the future, we plan to integrate our reinforcement learning method with the partitioning process. That is, in addition to optimizing over policies on the constructed finite-memory MDP, we would like to use the gained (implicit) information about the system's dynamics to update the boundaries of the partition. This may lead to policies that are more consistent with the underlying agent's dynamics.

References

Abate, A., D'Innocenzo, A., and Di Benedetto, M. (2011). Approximate abstractions of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 56(11):2688–2694.

Abbas, H., Hoxha, B., Fainekos, G., and Ueda, K. (2014). Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *2014 IEEE 4th Annual International Conference onCyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 1–6.

Agha-mohammadi, A.-A., Chakravorty, S., and Amato, N. M. (2011). Firm: Feedback controller-based information-state roadmap - a framework for motion planning under uncertainty. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4284–4291.

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102 – 114.

Angluin, D. and Fisman, D. (2014). Learning regular omega languages. In Auer, P., Clark, A., Zeugmann, T., and Zilles, S., editors, *Algorithmic Learning Theory*, volume 8776 of *Lecture Notes in Computer Science*, pages 125–139. Springer International Publishing.

Asarin, E., Donzé, A., Maler, O., and Nickovic, D. (2012). Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160. Springer.

Auslander, B., Gupta, K., and Aha, D. (2011). Comparative evaluation of anomaly detection algorithms for local maritime video surveillance. *Proceedings of the Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense X*.

Aydin Gol, E., Lazar, M., and Belta, C. (2012). Language-guided controller synthesis for discrete-time linear systems. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 95–104, New York, NY, USA.

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. (2013). On the robustness of temporal properties for stochastic models. *Electronic Proceedings in Theoretical Computer Science*, 125:3–19.

Bauer, A., Leucker, M., and Schallhart, C. (2011). Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology*, 20(4):14:1–14:64.

Bellman, R. E. (1957). *Dynamic Programming*. Dover Publications, Incorporated.

Belta, C. and Habets, L. (2006). Control of a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749 – 1759.

Belta, C., Isler, V., and Pappas, G. J. (2005). Discrete abstractions for robot planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874.

Bertsekas, D. P. (2000). *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition.

Binney, J., Krause, A., and Sukhatme, G. (2010). Informative path planning for an autonomous underwater vehicle. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4791 –4796.

Bourgault, F., Makarenko, A. A., Williams, S. B., Grocholsky, B., and Durrant-Whyte, H. F. (2002). Information based adaptive robotic exploration. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS 02)*, pages 540–545.

Brazdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretinsky, J., Kwiatkowska, M., Parker, D., and Ujma, M. (2014). Verification of markov decision processes using learning algorithms. In Cassez, F. and Raskin, J.-F., editors, *Automated Technology for Verification and Analysis*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer International Publishing.

Briers, M., Doucet, A., and Maskell, S. (2010). Smoothing algorithms for statespace models. *Annals of the Institute of Statistical Mathematics*, 62:61–89.

Candido, S. and Hutchinson, S. (2011). Minimum uncertainty robot navigation using information-guided POMDP planning. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6102 –6108.

Cassandras, C. G. and Lin, X. (2013). Optimal control of multi-agent persistent monitoring systems with performance constraints. In Tarraf, D. C., editor, *Control of Cyber-Physical Systems*, volume 449 of *Lecture Notes in Control and Information Sciences*, pages 281–299. Springer International Publishing.

Castro, P. S., Panangaden, P., and Precup, D. (2009). Equivalence relations in fully and partially observable markov decision processes. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1653–1658, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15.

Chen, Y., Ding, X., Stefanescu, A., and Belta, C. (2013a). A formal approach to deployment of robotic teams in an urban-like environment. In *Distributed Autonomous Robotic Systems*, volume 83 of *Springer Tracts in Advanced Robotics*, pages 313–327. Springer Berlin Heidelberg.

Chen, Y., Tůmová, J., Ulusoy, A., and Belta, C. (2013b). Temporal logic robot control based on automata learning of environmental dynamics. *International Journal of Robotics Research*, 32(5):547–565.

Choi, H.-L. and How, J. P. (2010). Continuous trajectory planning of mobile sensors for informative forecasting. *Automatica*, 46(8):1266–1275.

Cimatti, A. and Roveri, M. (2000). Conformant planning via model checking. In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning*, volume 1809 of *Lecture Notes in Computer Science*, pages 21–34. Springer Berlin Heidelberg.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. Wiley-Interscience, 2nd edition.

Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press.

Diaz-Mercado, Y., Jones, A., Belta, C., and Egerstedt, M. (2015). Correct-by-construction control synthesis for multi-robot mixing. In *2015 IEEE 54th Annual Conference on Decision and Control (CDC)*. Submitted.

Ding, X. C., Belta, C., and Cassandras, C. (2010). Receding horizon surveillance with temporal logic specifications. In *49th IEEE Conference on Decision and Control (CDC)*, pages 256 –261.

Ding, X. C., Lazar, M., and Belta, C. (2012). Receding horizon temporal logic control for finite deterministic systems. In *American Control Conference (ACC)*, Montreal, Canada.

Ding, X. C., Smith, S., Belta, C., and Rus, D. (2011). Mdp optimal control under temporal logic constraints. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 532 –538.

Ding, X. C., Smith, S. L., Belta, C., and Rus, D. (2014). Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257.

Dokhanchi, A., Hoxha, B., and Fainekos, G. (2014). On-line monitoring for temporal logic robustness. In *Runtime Verification*, pages 231–246. Springer.

Donzé, A., Fanchon, E., Gattepaille, L. M., Maler, O., and Tracqui, P. (2011). Robustness analysis and behavior discrimination in enzymatic reaction networks. *PLoS One*, 6(9):e24246.

Donzé, A. and Maler, O. (2010). Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer.

Fainekos, G. E. (2011). Revising temporal logic specifications for motion planning. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 40–45. IEEE.

Fainekos, G. E. and Pappas, G. J. (2009). Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291.

Farwell, J. P. and Rohozinski, R. (2011). Stuxnet and the future of cyber war. *Survival*, 53(1):23–40.

Fawzi, H., Tabuada, P., and Diggavi, S. (2012). Security for control systems under sensor and actuator attacks. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*, pages 3412–3417. IEEE.

Fu, J. and Topcu, U. (2014). Probably approximately correct mdp learning and control with temporal logic constraints. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA.

Gan, S. K., Fitch, R., and Sukkarieh, S. (2012). Real-time decentralized search with inter-agent collision avoidance. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 504 –510.

Gastin, P. and Oddoux, D. (2001). Fast ltl to bchi automata translation. In Berry, G., Comon, H., and Finkel, A., editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin Heidelberg.

Ghaffarkhah, A. and Mostofi, Y. (2012). Path planning for networked robotic surveillance. *IEEE Transactions on Signal Processing*, 60(7):3560 –3575.

Gol, E. A., Densmore, D., and Belta, C. (2013). Data-driven verification of synthetic gene networks. In *52nd IEEE Conference on Decision and Control (CDC)*.

Gupta, A., Langbort, C., and Basar, T. (2010). Optimal control in the presence of an intelligent jammer with limited actions. In *2010 49th IEEE Conference on Decision and Control (CDC)*, pages 1096–1101. IEEE.

Hoffmann, G. M. and Tomlin, C. J. (2010). Mobile sensor network control using mutual information methods and particle filters. *IEEE Transactions on Automatic Control*, 55(1):32–47.

Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201.

Jansen, D. N., Nielson, F., and Zhang, L. (2012). Belief bisimulation for hidden markov models - logical characterisation and decision algorithm. In Goodloe, A. and Person, S., editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 326–340. Springer.

Jin, X., Donze, A., Deshmukh, J., and Seshia, S. (2013). Mining requirements from closed-loop control models. In *Hybrid Systems: Computation and Control (HSCC)*.

Jones, A., Aksaray, D., Kong, Z., Schwager, M., and Belta, C. (2015a). Robust satisfaction of temporal logic specifications via reinforcement learning. In *54th IEEE Conference on Decision and Control (CDC)*. (Submitted).

Jones, A., Kong, Z., and Belta, C. (2014). Anomaly detection in cyber-physical systems: A formal methods approach. In *IEEE Conference on Decision and Control (CDC) 2014*.

Jones, A., Schwager, M., and Belta, C. (2013a). Distribution temporal logic: Combining correctness with quality of estimation. In *52nd IEEE Conference on Decision and Control (CDC)*.

Jones, A., Schwager, M., and Belta, C. (2013b). A receding horizon algorithm for informative path planning with temporal logic constraints. In *International Conference on Robotics and Automation (ICRA)*.

Jones, A., Schwager, M., and Belta, C. (2015b). Formal synthesis of optimal information-gathering policies. *IEEE Transactions on Robotics*. (Submitted).

Jones, A., Schwager, M., and Belta, C. (2015c). Information-guided persistent monitoring under temporal logic constraints. In *IEEE American Control Conference (ACC)*.

Julian, B. J., Angermann, M., Schwager, M., and Rus, D. (2012). Distributed robotic sensor networks: An information-theoretic approach. *International Journal of Robotics Research*, 31(10):1134–1154.

Julius, A. and Pappas, G. (2009). Approximations of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 54(6):1193–1203.

Kaelbling, L. P., Littman, M., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence in Medicine*, 101:99–134.

Kamgarpour, M., Ding, J., Summers, S., Abate, A., Lygeros, J., and Tomlin, C. (2011). Discrete time stochastic hybrid dynamic games: Verification and controller synthesis. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference*, pages 6122–6127.

Kassir, A., Fitch, R., and Sukkarieh, S. (2012). Decentralised information gathering with communication costs. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2427 –2432.

Kirby, J. R. (2010). Synthetic biology: Designer bacteria degrades toxin. *Nature Chemical Biology*, 6(6):398–399.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176.

Kloetzer, M. and Belta, C. (2008). A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287 –297.

Kong, Z., Jones, A., and Belta, C. (2015). Formal methods for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*. (Under review).

Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., and Belta, C. (2014). Temporal logic inference for classification and prediction from data. In *The 17th International Conference on Hybrid Systems: Computation and Control (HSCC)*, Berlin, Germany.

Kowalska, K. and Peel, L. (2012). Maritime anomaly detection using gaussian process active learning. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 1164–1171. IEEE.

Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299.

Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. (2007). Where's waldo? sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121.

Kupferman, O. and Vardi, M. Y. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19:291–314.

Kurniawati, H., Du, Y., Hsu, D., and Lee, W. (2011). Motion planning under uncertainty for robotic tasks with long time horizons. *Robotics Research*, 70:151–168.

Kurniawati, H., Hsu, D., and Lee, W. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*.

Lahijanian, M., Andersson, S., and Belta, C. (2012a). Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409.

Lahijanian, M., Andersson, S. B., and Belta, C. (2012b). Approximate markovian abstractions for linear stochastic systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 5966–5971, Maui, HI, USA.

Lahijanian, M., Andersson, S. B., and Belta, C. (2015). Formal verification and synthesis for discrete-time stochastic systems. *IEEE Transactions on Automatic Control*, (in press).

Lan, X. and Schwager, M. (2013). Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2415–2420.

Latvala, T. (2003). Efficient model checking of safety properties. In Ball, T. and Rajamani, S., editors, *Model Checking Software*, volume 2648 of *Lecture Notes in Computer Science*, pages 624–636. Springer Berlin / Heidelberg.

Leahy, K., Jones, A., Schwager, M., and Belta, C. (2015). Distributed informative path planning under temporal logic constraints. In *54th IEEE Conference on Decision and Control (CDC)*. (Submitted).

Luna, R., Lahijanian, M., Moll, M., and Kavraki, L. E. (2014). Asymptotically optimal stochastic motion planning with temporal goals. In *Workshop on the Algorithmic Foundations of Robotics*, Istanbul, Turkey.

Lygeros, J., Johansson, K., Sastry, S., and Egerstedt, M. (1999). On the existence of executions of hybrid automata. In *Proceedings of the 38th IEEE Conference on Decision and Control, 1999*, volume 3, pages 2249–2254 vol.3.

Madani, O., Hanks, S., and Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34.

Maler, O. and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 71–76.

Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition.

Meliou, A., Krause, A., Guestrin, C., and Hellerstein, J. M. (2007). Nonmyopic informative path planning in spatio-temporal models. In *Proceedings of the 22nd national conference on Artificial intelligence*, volume 1, pages 602–607.

Melo, F. S. Convergence of q-learning: a simple proof. http://users.isr.ist.utl.pt/ mtjspaan/readingGroup/ProofQlearning.pdf.

Monahan, G. E. (1982). A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):pp. 1–16.

Pasqualetti, F., Dorfler, F., and Bullo, F. (2011). Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design. In *2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 2195–2201. IEEE.

Pineau, J. (2004). *Tractable Planning Under Uncertainty: Exploiting Structure*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico. IJCAI.

Raman, V., Donze, A., Maasoumy, M., Murray, R. M., Sangiovanni-Vincentelli, A., and Seshia, S. A. (2014). Model predictive control with signal temporal logic specifications. In *Proceedings of IEEE Conference on Decision and Control (CDC)*.

Raman, V., Donz, A., Sadigh, D., Murray, R. M., and Seshia, S. A. (2015). Reactive synthesis from signal temporal logic specifications. In *The 18th International Conference on Hybrid Systems: Computation and Control (HSCC)*. To appear.

Rizk, A., Batt, G., Fages, F., and Soliman, S. (2008). On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Computational Methods in Systems Biology*, pages 251–268. Springer.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: a Modern Approach*. Prentice Hall.

Sadigh, D., Kim, E., Coogan, S., Sastry, S., and Seshia, S. (2014). A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *2014 IEEE 53rd Annual Conference on Decision and Control (CDC)*.

Salis, H., Tamsir, A., and Voigt, C. (2009). Engineering bacterial signals and sensors. *Contributions to Microbiology*, 16:194–225.

Scharf, L. and Demeure, C. (1991). *Statistical signal processing: detection, estimation, and time series analysis*. Addison-Wesley Series in Electrical and Computer Engineering. Addison-Wesley Pub. Co.

Schwager, M., Dames, P., Rus, D., and Kumar, V. (2011). A multi-robot control policy for information gathering in the presence of unknown hazards. In *Proceedings of the International Symposium on Robotics Research (ISRR 11)*.

Sen, K., Viswanathan, M., and Agha, G. (2005). On statistical model checking of stochastic systems. In *In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science*, pages 266–280. Springer.

Shani, G., Pineau, J., and Kaplow, R. (2012). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, pages 1–51.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656.

Shin, H. J., Eom, D.-H., and Kim, S.-S. (2005). One-class support vector machinesan application in machine fault detection and classification. *Computers & Industrial Engineering*, 48(2):395–408.

Shoukry, Y., Araujo, J., Tabuada, P., Srivastava, M., and Johansson, K. H. (2013). Minimax control for cyber-physical systems under network packet scheduling attacks. In *Proceedings of the 2nd ACM international conference on High confidence networked systems*, pages 93–100. ACM.

Simon, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience.

Singh, A., Krause, A., Guestrin, C., Kaiser, W., and Batalin, M. (2007). Efficient planning of informative paths for multiple robots. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2204–2211.

Sistla, A. P., Žefran, M., and Feng, Y. (2011). Monitorability of stochastic dynamical systems. In *Computer Aided Verification*, pages 720–736. Springer.

Slay, J. and Miller, M. (2007). *Lessons learned from the maroochy water breach*. Springer.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088.

Smith, S. L., Schwager, M., and Rus, D. (2012). Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426.

Smith, T. and Simmons, R. G. (2005). Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*.

Svorenova, M., Kretínský, J., Chmelik, M., Chatterjee, K., Cerná, I., and Belta, C. (2014). Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In *Hybrid Systems Computation and Control (HSCC) 2014*.

Teixeira, A., Pérez, D., Sandberg, H., and Johansson, K. H. (2012). Attack models and scenarios for networked control systems. In *Proceedings of the 1st international conference on High Confidence Networked Systems*, pages 55–64. ACM.

Thrun, S. (2008). Simultaneous localization and mapping. In Jefferies, M. and Yeap, W.-K., editors, *Robotics and Cognitive Approaches to Spatial Mapping*, volume 38 of *Springer Tracts in Advanced Robotics*, pages 13–41. Springer Berlin / Heidelberg.

Trevor, H., Robert, T., and Friedman, J. J. H. (2001). *The elements of Statistical Learning*. Springer.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202.

Ustun, B., Tracà, S., and Rudin, C. (2013). Supersparse linear integer models for interpretable classification. *arXiv preprint arXiv:1306.6677*.

Varakantham, P., Maheswaran, R., and Tambe, M. (2006). Implementation techniques for solving pomdps in personal assistant agents. In *Proceedings of the Third international conference on Programming Multi-Agent Systems*, ProMAS'05, pages 76–89, Berlin, Heidelberg. Springer-Verlag.

Vardi, M. Y. (1996). An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag.

Wolff, E. M., Topcu, U., and Murray, R. M. (2012). Robust control of uncertain markov decision processes with temporal logic specifications. In *IEEE American Control Conference*.

Wongpiromsarn, T., Frazzoli, E., Wongpiromsarn, T., and Frazzoli, E. Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In *2012 IEEE 51st Annual Conference on Decision and Control (CDC)*.

Wongpiromsarn, T., Topcu, U., and Murry, R. (2010). Receding horizon control for temporal logic specifications. In *HSCC10: Proceedings of the 13th ACM International Conference on Hybrid Systems:Computation and Control*, pages 101–110.

Yang, H., Hoxha, B., and Fainekos, G. (2012). Querying parametric temporal logic properties on embedded systems. In *Testing Software and Systems*, pages 136–151. Springer.

Zhang, L. (2004). Logic and model checking for hidden markov models. Master's thesis, Saarland University.

Zhang, L., Hermanns, H., and Jansen, D. (2005). Logic and model checking for hidden markov models. In Wang, F., editor, *Formal Techniques for Networked and Distributed Systems - FORTE 2005*, volume 3731 of *Lecture Notes in Computer Science*, pages 98–112. Springer Berlin Heidelberg.

Zhu, M. and Martínez, S. (2013). On distributed constrained formation control in operator–vehicle adversarial networks. *Automatica*, 49(12):3571–3582.

Zuliani, P., Platzer, A., and Clarke, E. M. (2010). Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 243–252. ACM.

Curriculum Vitae

Contact

Austin Jones

Division of Systems Engineering, Boston University, 15 St. Mary's St., Boston, MA 02215, USA

Email: austinmj@bu.edu

Areas of specialization

Robotics: single- and multi-agent systems

Formal methods for planning under uncertainty (partially observed systems)

Active sensing and optimal estimation

Data-driven inference for cyber-physical systems-

Experience

2011-2015: Graduate research assistant
Boston University, Boston, MA
Research advisors: Calin Belta and Mac Schwager
Research focuses on planning complex missions in which robots operate in partially observed environments that can be sensed via noisy, on-board sensors. Results in this area include synthesizing optimal information-gathering policies and developing a new framework for describing complex tasks in partially observed environments. Other topics include inferring cyber-physical system properties from execution data via novel algorithms that combine machine learning with formal methods.

2014: Summer research intern,
MIT Lincoln Laboratory, Lexington, MA
Worked in the Ballistic Missile Defense Systems Integration group on sensor management algorithms.

2010-2011: Research scientist,
Numerica, Loveland, CO
Developed, analyzed, and tested via simulation optimal detection, estimation, and data fusion algorithms.

Education

2011-2015: Ph.D. Systems Engineering,
Boston University, Boston, MA

GPA: 3.96

2006-2015 B.S.,M.S. Systems Science,
Washington University in St. Louis, St. Louis, MO
GPA: 3.60

Honors and Awards

2011: Dean's Fellowship, Boston University
Merit-based fellowship for incoming graduate students

2007-2010: Dean McKelvey fellowship
Fellowship for undergraduate researchers

2006-2010 Woodward Fellowship, Washington University in St. Louis
Merit-based scholarship for undergraduate engineering students

Skills

Scientific Programming
Python (SciPy, NumPy, NetworkX)
Matlab (Simulink)

Language
Proficient in written and spoken Spanish.

Publications

1. Austin Jones, Mac Schwager, and Calin Belta "Information-guided persistent monitoring under temporal logic constraints", IEEE American Control Conference (ACC) 2015.

2. Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta "SpaTeL: A Novel Spatial-Temporal Logic and Its Applications to Networked Systems" International Conference on Hybrid Systems: Computation and Control (HSCC) 2015.

3. Austin Jones, Zhaodan Kong, and Calin Belta "Anomaly Detection in Cyber-Physical Systems: A Formal Methods Approach", IEEE Conference on Decision and Control (CDC) 2014

4. Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta "Temporal Logic Inference for Classification and Prediction from Data", International Conference on Hybrid Systems: Computation and Control (HSCC) 2014

5. Austin Jones, Mac Schwager, and Calin Belta. "Distribution Temporal Logic: Combining Correctness with Quality of Estimation." IEEE Conference on Decision and Control (CDC) 2013

6. Austin Jones and Sean B. Andersson. "A Motion-Based Communication System.", IEEE American Control Conference (ACC) 2013.

7. Austin Jones, Mac Schwager, and Calin Belta. "A Receding Horizon Algorithm for Informative Path Planning with Temporal Logic Constraints.", IEEE International Conference on Robotics and Automation (ICRA) 2013.

8. Austin Jones, Mac Schwager, and Calin Belta "Formal Synthesis of Optimal Information-Gathering Policies", IEEE Transactions on Robotics (T-RO), Under Review

9. Kevin Leahy, Austin Jones, Mac Schwager, and Calin Belta "Multi-agent informative path planning subject to temporal logic constraints", IEEE Conference on Decision and Control 2015, Under Review

10. Yancy Diaz-Mercado, Austin Jones, Calin Belta, and Magnus Egerstedt "Correct-by-Construction Control Synthesis for Multi-Robot Mixing", IEEE Conference on Decision and Control 2015, Under Review

11. Austin Jones, Derya Aksaray, Zhaodan Kong, Mac Schwager, and Calin Belta "Robust Satisfaction of Temporal Logic Specifications via Reinforcement Learning", IEEE Conference on Decision and Control 2015, Under Review

12. Zhaodan Kong, Austin Jones, and Calin Belta "Formal Methods for Learning and Detection of Anomalous Behavior", IEEE Transactions on Automatic Control (TAC), Under Review